RL-TR-95-209, Vol II (of two)
Final Technical Report
October 1995

PARALLEL SOFTWARE ENGINEERING TECHNOLOGY FORECAST, Assessment, Trends, Vision, and Strategy

RCI, Ltd.

Carl Murphy

Rome Laboratory
Air Force Materiel Command
Rome, New York

RL-TR-95-209, Vol II (of two), has been reviewed and is approved for publication.

APPROVED: Joseph P Cavano

JOSEPH P. CAVANO
Project Engineer

FOR THE COMMANDER: [illegible]

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE October 1995	3. REPORT TYPE AND DATES COVERED Final Jul 94 - Jun 95

4. TITLE AND SUBTITLE
PARALLEL SOFTWARE ENGINEERING TECHNOLOGY FORECAST, Assessment, Trends, Vision, and Strategy

5. FUNDING NUMBERS
C - F30602-94-C-0108
PE - 62702F
PR - 5581
TA - 20
WU - PH

6. AUTHOR(S)
Carl Murphy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
RCI, Ltd.
1301 East 79th Street, Suite 200
Minneapolis MN 55425

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory/C3CB
525 Brooks Rd
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-95-209, Vol II (of two)

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Joseph P. Cavano/C3CB/(315) 330-4063

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Rome Laboratory developed a "Parallel Software Technology Forecast" to identify the parallel software engineering technology that will be required to meet Air Force needs for mission-critical software in a High Performance Computing environment for the next decade. It concentrated on the quality and cost issues of software development for Command, Control, and Communications (C3I) applications and addressed the following goals: (1) anticipate technology directions of the parallel computer industry and forecast parallel software technology capabilities; (2) identify key C3I factors in the Air Force and show what the implications of HPC might be on the Air Force's ability to develop productive and efficient C3I applications software; and (3) compare and contrast Air Force needs for parallel software technology to that in the commercial sector.

Rome Laboratory assembled a distinguished Blue Ribbon Panel consisting of seven technical experts, and solicited position papers from a broader range of position makers from academia, government, and industry.

14. SUBJECT TERMS
Parallel software engineering, Parallel processing, Parallel software development

15. NUMBER OF PAGES
134

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form [illegible]
Prescribed by ANS [illegible]
298-102

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ANNEXES

1. PREFACE

Rome Laboratory developed this technology forecast to identify the parallel software engineering technology required to meet Air Force mission-critical needs in High Performance Computing (HPC) environments over the next decade. The forecast concentrates on the quality and cost issues of the software development process for Command, Control and Communications applications (C^3I) as well as performance. The "Parallel Software Technology Forecast" addresses three goals:

1) Anticipate technology directions of the parallel computer industry and forecast parallel software technology capabilities, ie, current trends in software technology related to emerging trends in HPC.

2) Identify key C^3I factors in the Air Force mission and show what the implications of HPC are for the ability of the Air Force to develop C^3I application software productively and efficiently.

3) Compare and contrast Air Force needs for parallel software technology in the commercial sector and identify opportunities for effective bi-directional technology transfer.

The technology forecast involved five stages:

1) Rome Laboratory produced a condensed Parallel Software Engineering Technology Assessment (RL TR).

2) Rome Laboratory assembled a distinguished Blue Ribbon Panel consisting of technical experts from industry and academia. The Panel included:

Dr. Walter Beam (noted author on C^3I and system engineering)
Mr. C. Gordon Bell, (Chairman, University Video Communications)
Prof. Tom Cheatham (Harvard University)
Dr. George Lindamood (Director of Information Services for the State of Washington)
Dr. Jeffrey Mohr (Chief Scientist, Information Technology Solutions)

3) Rome Laboratory announced a Technology Forecast Forum to solicit position papers. A broad range of *position makers* from academia, government and industry responded. Twenty positions were selected for review by the Blue Ribbon Panel as representative of the state of the art.

4) Rome Laboratory sponsored an invitation-only, Blue Ribbon Technology Forecast meeting at the Orlando Hyatt on January 23-26, 1995. Panel members and position makers interacted to produce the technology forecast. Presentations of each participant's view were followed by separate sessions on C^3I, commercial off-the-shelf, and parallel technology impacts. The group then collaborated on notes that represent the combined trends, vision and strategy for C^3I parallel software engineering.

5) After the interaction, this author set down the conclusions of the panel and distributed it for confirmation by the Panel Members. The reports are:

"Volume I: C^3I Parallel Software Engineering Technology Forecast -- Conclusions (draft RL TR)"

"Volume II: C^3I Parallel Software Engineering Technology Forecast -- Assessment, Trends, Vision and Strategy"(draft RL TR)

Volume II incorporates all material from Volume I. It also provides an assessment, analysis and confirmation of the Blue Ribbon Panel's Conclusions.

2. EXECUTIVE SUMMARY

2.1 C^3I Parallel Software Engineering Strategy

The panel established clear directions for parallel software engineering processes and infrastructure. The discussion mechanisms of high-time-value, tough character applications and C^3I mission integrity factors express the challenges of parallel software engineering. (See Insert A.) C^3I software must deliver all these elements at once, along with performance prediction at system definition, design and implementation layers. This is a daunting task for a higgledy-piggledy market mixture of architectures. (Higgledy-piggledy expresses the drastic performance differences found for general applications on massively parallel architectures.) However, the panel set the architectural target as a mix of high bandwidth capability distributed nodes and low latency multiprocessors. This means that parallel software engineers can separately address message passing applications and low-latency applications. They need tools to interface and control the interaction between them. With simplified architectures the present day's mapping issues can become more standardized. Now they can turn their attention toward performance prediction and effectiveness, by using parallel software engineering to achieve C^3I mission integrity factors. Their tools and mechanisms must address combinations of high bandwidth capacity networks-of-nodes and low latency multiprocessors.

Throughout this report the following C^3I Parallel Software Engineering terms apply:

Mission Function or *Tough Character Applications* -- these combine a mix of high complexity, unstructured (state and data dependent) paths, high dynamic resource fluctuation and frequent synchronization characteristics found in C^3I environments.

Mission Integrity Factors or *High-Integrity-Value* -- these are fault tolerance, availability, graceful degradation, security, reliability, survivability, repairability, usability, twenty-four-hour, seven-day operation time frame and other factors that are required for the mission in combative environments.

Mission Performance and *High-Time-Value* -- C^3I applications operate where time responses are critical to mission success. Time is the essence of C^3I systems.

Parallel Software Engineering -- C^3I systems must be engineered and are typically built by large groups. C^3I parallel software engineering is the collection of processes and methods used to achieve predictable performance, programmability, portability, maintainability, adaptability and low life cycle costs.

Insert A -- Function, Integrity, Performance and Engineering

The panel also established criteria to govern engineering of C^3I systems. The software engineer must empirically investigate and understand the justification-decision boundary between choices of commodity-priced COTS, modular board level COTS, and military-unique components. They must devise the interfaces between these diverse technologies. They need incremental building methods for evolutionary build and test methods. Also necessary are probes and information gathering methods to take advantage of the "daily" exercised and evaluated C^3I system necessary to "make the warfighter the informed customer." Refinement layers that simplify change for new mission needs are another necessity. These must be in concert with rapid application development methods.

In summary, to meet these goals, parallel software engineering researchers must define new relationships with COTS computer vendors and integrators to figure out the boundary of COTS suitability. We must build relationships with COTS system vendors to establish testbed demonstrations of interfaces. Also, we must build different relationships with operating C^3I organizations to gather opinions on operations, testing and effectiveness on a shorter time scale.

2.2 Positions

2.2.1 C^3I Parallel software engineering

The positions of the Blue Ribbon panel members and position makers provided a vertical parallel software engineering structure. (See the Preface.) The positions gave an overview of the complex and dynamic needs of C^3I systems. They pointed out the relentless factor-of-four for every third-year growth in microprocessor-based computer and network technologies. The panel suggested that the highest performance COTS machinery would meet many needs of future C^3I missions. One position suggested was that there be a match between C^3I decision making needs and large scale business ones. They believed that parallel methods of collaborative interaction were another future C^3I software application. They recognized the need for globally or capability scalable architectures and noted the weak support for C^3I by parallel scientific methods.

2.2.2 An overview of parallel software engineering research

The positions described research on multi-level refinement techniques, virtual machine methods, components, compilers, debugging tools and run-time methods of parallel software engineering. The panel noted acceptance of virtual shared object models, debugging tools, large message passing virtual machines and the Bulk Synchronous Parallel (BSP) super-step synchronization model. However, the panelists also said that engineers have not accepted parallel methods into their processes. They found that there are too few independent software vendors using these techniques. The only widely accepted parallel application is the database management system. They thought that research described by the positions is primarily directed toward scientific research. It is not adequate for solving the high-time-value, tough application character and mission integrity factors necessary in C^3I systems.

2.3 Assessment

Time is the essence of C^3I *systems, but the mission integrity factors let the* C^3I *system provide decision making information to the warfighter during combat.*

The panel found that combative missions generate C^3I system's tough character applications, and require C^3I mission integrity factors. It found that today's government-funded scientific research parallel technology falls short of meeting C^3I mission needs. Instead, C^3I systems need a more general parallel capability to meet all C^3I character types and mission factors. In addition, the limited suitability of these machines makes system building and life-cycle support expensive and risky. Program managers expect that parallel software engineers will deliver systems and components with predictable time response and meet C^3I mission factors in a well-planned organized design and development process. Parallel software engineers need research directed specifically toward future C^3I systems if they are to meet C^3I expectations.

2.4 Trends

The panel concluded that C^3I systems would increase demands for high-time-value, tough application character and C^3I mission integrity factors. Parallel techniques could provide mechanisms to build C^3I mission factors like reliability, availability, security, fault tolerance and graceful reaction or degradation during harsh combative conditions. C^3I's tough application character renders large message passing techniques of scientific research ineffective. Data flow methods may be necessary to meet high-time-value C^3I application needs. *The panel believed that* C^3I *applications with shorter time scales, dynamic* *reconfiguration, complex data interaction, visualization demands, and collaborative interaction would grow in importance.* The panel also believed that interfaces and information interchange would become a critical technology for success of parallel computers in C^3I systems.

The panel projected a five-year trend for commercial parallel computing systems. Its commercial model is a network of nodes, servers and high bandwidth capacity interconnection. The architecture includes the following: nodes with a future Pentium microprocessor, Windows NT operating system, and ATM-based distributed network. Client programmers would reach all data and multimedia by SQL. This distributed system would have no shared memory. It includes only commodity hardware, data base and operating system software. The client process would view the network as a massive server, thus precluding the need for any large servers. The largest servers would have four processors. The panel justified its projection based on technical capability increases for single microprocessors (to 1 GMIPS), memory (to 256 MB), disk units (to 20 GB) and network bandwidth capacity (to 622 mbps).

The high bandwidth capacity of the network makes it effective for almost all of today's successful parallel throughput applications. These are transaction, decision support, replicated engineering and message passing scientific research. However, the panel believed that "pockets" of high-time-value, tightly coupled computing would retain a viable market. *This gives an architectural separation between a distributed COTS, high-bandwidth-capacity fabric and a low-latency, high-time-value multiprocessor.* This dichotomy separates bandwidth-only from low-latency demanding applications. Interfaces to legacy data systems would also influence these designs. In addition, the panel believed that commercial system builders might adopt object-oriented technology instead of the model's SQL as its lowest common data access denominator. (One potential solution, SQL3 combines the two.) Strategic decision making and concurrent engineering are reasons for the need for pockets of high-time-value and object-oriented methods.

2.5 Vision

The panel believed that the upgraded COTS fabric might not meet all C^3I demands. Possible shortfalls might be:

- ineffectiveness of SQL for diverse accessed and complex data type C^3I data
- over specialization of commercial network and supporting protocol designs for multimedia -- leaving the optimization of ATM bandwidth capacity for data transfer in a less than optimum state
- short fall of object-oriented methods in delivering on promises of design efficiency, performance and code reuse on large projects

To supplement these shortfalls, C^3I *systems would also need pockets of tightly coupled parallel computers. These needs are the following: high-time-value, diversity of applications with distributed data repositories, multiple access mechanisms, dynamic resource allocation, quick visual and imagery processing, necessity to meet* C^3I *integrity factors with predictable performance, and tough application character.* The reasonable expectation was that a COTS fabric, like the commercial one but built of the highest capability nodes instead of commodity ones, would serve many C^3I application needs. The panel suggested that the alternative of a single, general-purpose, parallel architecture that met all C^3I needs was out of reach in the present computer research and development environment.

2.6 Strategy

There will be no "silver bullet" that solves these challenges to C^3I *parallel software engineering. However, the panel outlined a path toward gaining the necessary foundation and knowledge to do timely and responsive* C^3I *engineering.*

Future parallel software engineers must meet mission constraints using commercial off-the-shelf (COTS) hardware and software wherever possible. A parallel solution is often necessary to the engineering problems created by the complexity of "high-time-value," C^3I mission factors and "tough" application character. The lack of any of these elements may limit the combative (competitive) capability of C^3I systems of the future. One threat is that *COTS distributed technology might allow third-world countries to inexpensively assemble a* C^3I *data fusion and decision capability*. Parallel software engineering methods are one way to maintain the Nation's command advantage.

The system designer should base designs on an architecture similar to the COTS fabric and must justify any parallel supplement. The panel set the following goals for parallel software engineering research:

- generate C^3I system design tools that give the following: 1) reconcile the COTS fabric and pockets of parallelism dichotomy, 2) predict performance, 3) establish system engineering metrics and 4) define the justification line between use of parallel and the COTS distributed fabric
- develop interface mechanisms to: 1) connect parallel plus COTS fabric architecture into C^3I systems, 2) achieve C^3I mission factor capability and 3) test and monitor C^3I exercises to capture mission factor and time value information
- find methods to 1) achieve dynamic reallocation of resources for meeting mission needs, 2) respond to collaborative interaction, 3) insert test or monitor probes for software engineering and recursion testing and 4) predict the scalability of applications in the C^3I environment
- reduce the cost and effort for development and life cycle support by 1) developing "architectural independent" design and programming models, 2) insisting on refinement and performance prediction at each level (system, design and programming) and 3) defining architectural independence as "generational portability" of today's solution to tomorrow's computer (generational portability refers to portability to architectures that are not yet built)
- interface and cooperate with COTS industry to define and continuously improve the COTS foundation fabric for C^3I use; experiment with COTS technology to enable continuous tracking of key commercial technologies, such as the following: operating systems, languages, SQL, object-based data and design techniques, visual programming, and high performance information transfer
- develop a demonstration testbed for emulating COTS architectures and add in the necessary coordination languages, effective protocols, and hardware interface drivers to show system functions.

3. STRATEGY -- Parallel Software Engineering

This section gives the strategy recommendations of the Blue Ribbon Panel. The assessment, trends, and vision of the panel along with confirming analysis and discussion are provided in the chapters that follow.

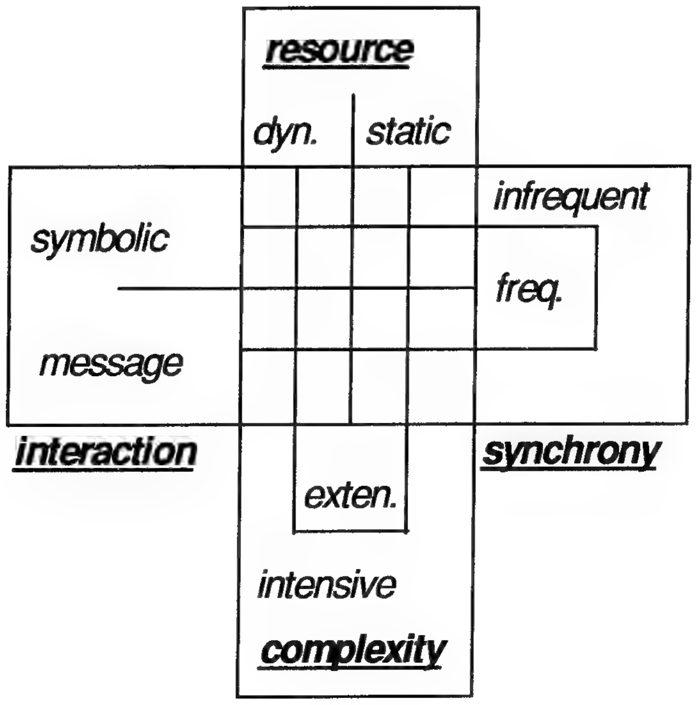
3.1 Strategic directions for C^3I parallel software engineering

The panel gave clear direction toward improving parallel software engineering processes and infrastructure. These directions are well supported by DOD sources, independent technology forecasts, and recent commercial publications.

3.1.1 Discussion mechanisms

The panel established the links between C^3I parallel software engineering and the C^3I mission requirements by stressing the terms time response (high-time-value), C^3I mission integrity factors (high integrity value) and C^3I application difficulty (tough character application). High time value is a label for applications requiring low-latency interaction. This is contrasted with large message passing where the importance is placed on high bandwidth capacity. These labels distinguish the value of time to the application and relate it to the architecture. Tough character application means complex, dynamic, frequent synchronization or communication and human and data dependent control paths. (See Figure 3-1.) This distinguishes the C^3I application from parallel scientific research computing and from the constant stream signal and image data collection flow applications. The C^3I mission factors examples are reliability, availability, graceful reaction or degradation, security, and physical constraints in equipment size, volume, power consumption, and electromagnetic radiation. These have a high integrity value in C^3I systems.

Figure 3-1. C3I Application Character



3.1.2 Architectural target

The COTS network-of-nodes architecture (built from high-end components) with added pockets of low-latency multiprocessor computing establishes a target architecture in which the needs of parallel software engineering can be met. Since it will evolve to match the latest commercial best practice its configuration is general and multipurpose. Yet it defines the environment that parallel software tools must target. The dichotomy of high bandwidth capacity and low latency existent in the same system gives just two target architectures for the tools and applications. Design refinement methods can now be targeted to just two architectural models. The same two targets can be used for compilers. Once placed in the category of low-time value, components need to target only a single architecture, the SNAP-like COTS one. Components with high-time-value are targeted only to low latency, globally scalable multiprocessor architecture. High integrity value applications must be evaluated to determine if the extra resources required, or the dynamic reaction scheme requires high-time-value or low time value.

Parallel software engineering tools can separately address applications suitable for message passing, applications requiring low latency, and the tools needed to interface and control the interaction between them. With these simplified architecture models the present day's mapping issues can become more standardized. Now the attention can turn toward performance prediction and effectiveness, use of parallel software engineering as a tool to achieve C^3I mission factors and on tools and mechanisms that work for one or the other of high bandwidth capacity networks of nodes and on low latency multiprocessors. The parallelism of both architectures can be used to provide some C^3I mission factors, eg, automatic generation of multiple tasks for the same task, with use of common voting or other techniques to obtain fault tolerance. Distributed methods as well as low latency ones would be developed depending upon the value of time in the application. Other C^3I factors could be achieved through parallel innovations in security, graceful reaction to unexpected events, and planned reaction to expected ones.

3.1.3 Parallel software engineering process goals

3.1.3.1 Simplification of parallel coding

With only two architectural types to target, a common machine model could be provided. The use of superstep barriers, as found in the Bulk Synchronous Parallel model, has been found to allow correctness proofs and predict performance. It also simplifies coding of complex applications. Restriction to complete all operations on a regular time frame is a common tool for meeting real time constraints in C^3I systems. BSP gives a logical time frame for programming of complex and dynamic applications.

3.1.3.2 COTS and military-unique decision surface

The panel also laid out some parallel software engineering goals that affect the way C^3I systems would be engineered. The justification decision boundary between choices of commodity-priced COTS, modular board level COTS, and military-unique approaches are to be investigated and understood. Interfaces between these diverse technologies are required to be engineered and converge. Incremental building methods are required to meet the evolutionary build and test methods. Probes and information gathering methods are indicated to take advantage of the "daily" exercised and evaluated C^3I system necessary to "make the warfighter the informed customer." Engineering refinement layers are needed to facilitate change and to meet new mission needs. These must be in concert with rapid application development methods.

3.1.3.3 Interfaces

In summary, to meet these goals parallel software engineering researchers will have to define new relationships with COTS computer vendors and integrators to determine the boundary of COTS suitability. They must build relationships with COTS system vendors to establish testbed demonstrations of interfaces.

They also need bi-directional relationships with operating C^3I organizations to gather feedback on operations, testing and effectiveness on a shorter time scale.

3.1.3.4 Incremental development, frequent interaction and collection of information

The best commercial practice is to have many incremental deliveries on the way to reaching a stable system. This method ensures that the information infrastructure of the organization is prepared to operate and maintain a system. C^3I methods recommended by the panel suggest a similar approach to meet the short development time frames and constant interaction with operational and warfighting elements.

3.2 Planned research focus areas

The panel believed that the goal of parallel software engineering was to meet both high-time-value goals and C^3I mission factors within the parallel environment. Applications of any character should be feasible with performance, programmability, and portability. In order to accomplish this daunting task, the panel also believed that parallel software engineering must concentrate on the most difficult character (extensive complexity, dynamic resource demands, symbolic interaction and frequent synchrony), ie, the toughest problem types. (See position 2,2 in Figure 1.) However, parallel software engineering must recognize the need to integrate those results with existing C^3I systems and the vision of commodity SNAP-like architectures.

The vision of the forecast panel for parallel software engineering concentrates on how to make parallel computing effective for C^3I systems development. The panel expressed this concept in several ways during the course of the working session.

3.2.1 Support for dynamic adaptation

The panel stated that a critical capability was one that let the system rapidly and effectively change the direction of processing. The system makes these rapid changes to respond to human interaction, mission change, and new phases of operation. The system can change mission activity to respond to better information. This may be one of the valuable assets of parallel computing when resources switch to meet the newly demanded processing. Dynamic adaptation based on decisions made by the C^3I system would require parallel computing for artificial intelligence and knowledge-based decision making. In addition, parallelism and dynamic adaptation allow the system to provide fault tolerance and resiliency by determining its state and responding with a reconfiguration that meets mission needs to the best of its capability. Parallel software engineering researchers should generate an application framework for creating dynamic resource changes in parallel systems.

3.2.2 Architecture independent software representations

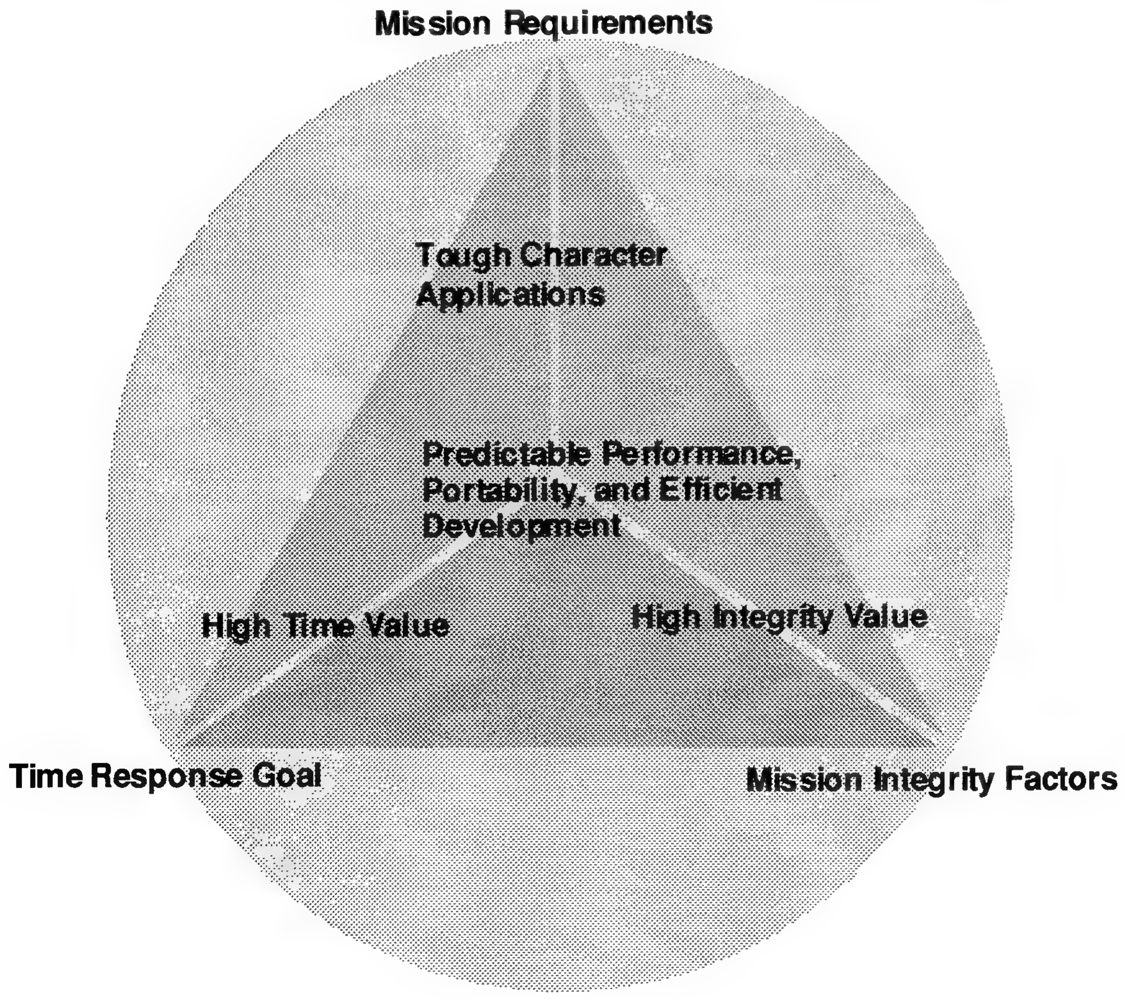
The industry needs an architecture independent programming model. Defining such a model has many difficulties because of the diversity of application character and architectural capability. The panel and position makers described several approaches that built a hierarchy of layers of design, architectural selection and then machine selection. In addition, automatic parallel program generators for these layers are the subject of research of several position makers. However, none had applied these methods to the tough application character type problems that are similar to C^3I ones. A problem seems to be that the architectural targets have been parallel computers and tools built primarily for scientific research. The panelists believed that, due to the significant difference in character and requirements, that a simple transfer of this technology from scientific research might not be effective. This does not lessen the need for the hierarchical tools. However, it does mean that their target should be the simple dichotomy of message passing or low latency multiprocessor. Researchers should provide architectural independence so that C^3I can apply the latest processor and architecture capabilities without the need to continually port to new architectures. This.generational portability is critical to maintaining military advantage. In addition, researchers should define their refinement hierarchy and capability to allow the removal of the uncertain availability of specific processor architectures.

3.2.3 Parallel software engineering tools for the C^3I system

The panel believed that C^3I engineers need tools at a system level scope to deal with engineering of parallel software in C^3I systems. Engineers need processes and tools for design modeling of parallel computer hardware and software in a hierarchically staged process. The tools should be focused on the tough applications that C^3I systems require. Researchers should develop a hierarchy of executable specification tools for high-time-value design and development layering.

Engineers also need performance prediction tools. These tools should be in a hierarchy of design stages. At each stage an engineer should be able to predict performance and cost and evaluate suitability of programming models for parallel architectures. These tools must resolve the time uncertainty problems of parallel computers. Such tools should give design visibility at each level and provide reverse engineering capability for better understanding of the allocation of components to time response, C^3I mission factors and C^3I requirements.

Figure 3-2. The Parallel Software Engineering Challenge


Mission Requirements
Tough Character Applications
Predictable Performance, Portability, and Efficient Development
High Time Value
High Integrity Value
Time Response Goal
Mission Integrity Factors

3.2.4 Tools to evaluate and apply COTS hardware, software and tools to C^3I applications

Researchers should track commercial advances in distributed, coordination, operating system, data or object base and language software. They should adopt these into the C^3I system model where there is a benefit. This is reasonably expected to be the least expensive manner to fulfill a majority of the mission needs.

To facilitate this research, they need to interface and cooperate with industry to achieve an inexpensive but effective SNAP-like architecture as a common fabric for C^3I systems. Knowing that the common fabric will work requires that engineers do an advanced emulation of the target architectures. The test cases should be tough character C^3I applications and include mechanisms for obtaining C^3I mission integrity factors like reliability, fault tolerance, security, graceful degradation, and dynamic resource change.

The C^3I development tools should be integrated with COTS SNAP-like architectures. These system engineering tools provide the component allocations to SNAP-like COTS systems combined with multiprocessors. The activity of integration should be supported by processes for designing and implementing effective interfaces from parallel to other C^3I system components. Concentration should be placed on clusters of nodes that represent the SNAP-like COTS architectures. Researchers also need to create system supervisor/ control mechanisms and the necessary additions to commodity COTS for control of the SNAP-like architecture and interfaces to high-time-value nodes.

Because the panel anticipated certain areas of potential COTS shortfall, the integration research needs to pay attention to the capabilities of COTS workstations, local area networks, wide area networks and telephone switches using high bandwidth ATM switches. In addition, they should determine a suitable organization of C^3I data that best uses either SQL relational organization or object database organizations. Finally, the promise of object oriented programming methods must be tracked to see if their promise extends beyond the initial successes of graphical user interfaces in client-only programming.

3.2.5 Combine tough character application, high-time-value and high integrity value

The parallel research community needs to find methods of achieving high-time-value performance and gaining C^3I factors in a tough character application. (See **Figure 3-2**.) The community needs to develop tools that predict performance and factors on any mission character application. This needs to be accomplished while providing rapid software development, easy maintenance or enhancements, scalability across a wide range of mission scope, and portability from architectural generation to generation. Tools that reduce the risk of achieving these results on C^3I applications are extremely valuable. Tools that do not consider such complexity are of little value. Another approach suggested by the panel is to evaluate operating system, interconnect access protocols, and network protocols that limit parallel performance, restrict C^3I mission factors, reduce portability, prevent reuse and preclude performance prediction.

3.2.6 Integration to establish C^3I mission requirements for parallel components

The panel believed that parallel software engineering would be most valuable if more closely associated with exercises and demonstrations of C^3I systems. A constant improvement of the C^3I process could be facilitated if the software could be continuously improved. The recommendations were to attain tools and processes for the non-intrusive entry and coordination with C^3I systems to solve any refinement needs. Researchers should build tools that collect information to be used to create better designs and to refine the use of parallel computing in C^3I systems.

A needed tool is one for on-line monitoring of the C^3I environment that captures C^3I mission factors and time response values. Researchers need to create on-line monitoring and testing tools to reduce uncertainty of correctness, performance, and mission factors in the C^3I environment. These tools are also needed for determining capabilities and resources required to meet the mission time response and factor goals.

3.3 Analysis of new C^3I directions

A key result of a COTS strategy is that hardware is not preselected, but is put in action in a just-in-time manner. This is consistent with the Defense Studies Board (DSB) recommendation to align life cycle and adapt commercial practices [EE Times, Nov. 24, 1995]. The Blue Ribbon Forecast Panel did not discuss this impact on parallel software engineering in its recommendations. However, its recommended technology focus areas lead directly to supporting this feature when the system technology is COTS. It does mean that *the architecture-independence goal is of critical importance for the new software procurement paradigm.* The just-in-time hardware selection also gives more meaning to the need for understanding the interface and discriminators between C^3I military-unique and COTS components, understanding and performing a system design that is inclusive of parallel components that meet high-time-value requirements. This important need increases the need for tools and refinement process parallel software tools for adding parallel computers and software to the COTS SNAP fabric for high-time-value needs.

3.3.1 Potential for COTS use in C^3I

Using the Warfighter and DOD directions as a guide leads to a C^3I system of the future that consists largely of COTS components and software, whose software was developed ahead of or concurrently with its hardware. However, commercial practices and tools are one way to accomplish this strategy. Some problems can be anticipated.

- First, C^3I systems require significantly shorter response time frames than commercial information systems, have much higher complexity of data types and interactions, and have need for significantly stronger performance in graceful degradation, fault tolerance, security, reliability, and time guarantees. These C^3I mission integrity factors are more stringent than in commercial information decision systems. The C^3I character is also tougher than that of commercial information systems. However, they do have a degree of similarity because commercial information systems have requirements in each of these C^3I mission factors. Some transaction processing systems have a very high fault tolerance need compared to scientific research software where the effect of an occasional failure may only be an inconvenience that might be overlooked. However, the degree of these requirement is higher in C^3I systems.

  A second practice is that commercial firms do little application software development beyond integration of purchased independent software packages. For example, in the presently popular client/server architectures the applications are confined to client desktop graphical user interface programs that operate within a single workstation. Business logic, data base access, network management, replication transfer from legacy mainframes, and any other operations are expected to be built upon purchased packages. Unfortunately there are many products with confusing capabilities making it difficult to assemble a good system. Very few applications run across all these combinations [ComputerWorld, p 4, February 20, 1995]. In addition, key components are often missing and large teams must be assembled to fill in the missing components. System design tools are immature, leading to risky projects. This commercial process often leads to highly inefficient and chaotic commercial information systems that remain tied to expensive mainframes. Table 3-A shows that early adopters were mesmerized by hardware cost savings and over-optimistic estimates of cost savings.

TABLE 3-A Benefit shortfall (based on 305 IS managers; BRG Newton, MA)

Benefit area	Expected	Achieved
More flexibility	71%	51%
Increased productivity	66%	53%
Improved customer service	58%	39%
Decreased overhead	50%	31%
Less maintenance	46%	28%
Work force reduction	28%	16%
Increased market share	19%	10%

In reality, the commercial client/server marketplace has unfulfilled requirements similar to those of parallel software engineering in C3I systems. COTS technology does not automatically overcome the barriers to getting to straightforward system design, design visibility, ease of interaction and collaboration, standard interfaces, and necessity to become intimate with C3I applications, This result indicates that a software engineering capability needs to be built for software system design, interfaces, and application evaluation. The capability is required in both military and commercial development.

4. BACKGROUND

4.1 Overview of Blue Ribbon Panel

The parallel software engineering forecast is based on a parallel computing technology assessment, recommendations of the blue ribbon panel working session and supporting analysis. Panelists were provided a condensed version of the Rome Laboratory Parallel Software Engineering Technology Assessment (draft RL TR). A condensed version is provided as Annex A. Positions were solicited from a broad spectrum of academe and industry. Summaries of those positions are given in Annex B.

4.2 Overview of participants

The Forecast Panel for Parallel Software Engineering convened to seek out the expert opinions and forecast of leaders in C^3I related parallel technology. The panel included C^3I advisors, commercial, government, and academic members. The panel organizers selected them due to their cognizance of architectures and applications, components and markets for parallel systems. In order to provide the panel with a diverse set of ideas on parallel software engineering, the organizers assembled position makers to obtain their ideas and projection of the future of parallel software engineering. Position makers represented a wide range of activities, among which were the following:

- industry level -- machine cost models, research center activities, industry health status and future industry viewpoints
- system level -- system engineering, engineering applications, C^3I systems, and dynamic control
- programming models -- virtual machines and automated parallel programming
- programming tools -- visual, components, libraries, and debugging
- system effectiveness -- domain specific objects, libraries, and hardware architecture target development tools

The forecast experts focused their attention on C^3I system requirements for the following:

- high-time-value applications in C^3I systems
- C^3I mission factors (fault tolerance, security, graceful degradation, physical constraints, etc.)
- difficult or "tough" application character (dynamic resource demand, symbolic interaction, frequent synchronization, and extensive complexity)
- software engineering issues necessary for C^3I-like projects: programmability, rapid development, and maintenance, along with effective performance and portability with predictable design and development performance allocation

4.3 Goals and limits

This document's purpose is to assess parallel software engineering technology, define trends, describe the Blue Ribbon Panel's vision, and define a strategy for improving parallel software engineering. This document reports the full assessment, results of the panel working session, following concurrence, analysis, and confirming references. It concludes with a rationale for technology status, trends, and vision. It also provides Annex B -- "Blue Ribbon Forecast Panel Working Session and Position Summary," which summarizes position makers inputs. An executive level document "Parallel Software Engineering

Technology Forecast - Conclusion" (Rome Laboratory TR) gives the conclusions of the working session. The executive level document is incorporated in various sections of this document.

4.4 Scope, definition & assumptions

The panel set boundaries on the technology and forecast elements that it was to consider. The panel believed that the opportunities of parallel computing should derive from the mission needs, not from zealous enthusiasm over a particular technology. However, imposing this discipline should not inhibit the invention of new (or improvement of old) C^3I applications and capability. The panel stressed the importance of the potential of parallel technology to meet high-time-value C^3I component needs.

4.4.1 C^3I boundaries

To concentrate on the right issues, the panel defined the boundaries of C^3I systems as the following functions necessary to conduct Air Force warfare:

- data and information fusion
- information generation, display and visualization
- decision making
- human interaction with the process of command, control and intelligence
- action and reaction control
- warfare planning, direction, evaluation, and awareness
- managing the wide range of information that converges on command centers

Therefore, C^3I systems are planning, information and decision making elements of Air Force Warfare. The panel excluded certain subsystems, eg, sensor and image data sources for the C^3I system and the external weapons and delivery systems that take commands to respond to a threat or meet mission goals. (These are defined as effectors.) **Figure 4-1** provides a simple view of this boundary, where sensors and effectors are integrated by the C^3I system.

4.4.2 Commercial-off-the-shelf boundaries

The panel considered that one of its important roles was to determine the directions and condition of the commercial-off-the-shelf (COTS) parallel industry. The forces of change in commercial technology are the driving forces in computing. Also the economics of building military systems require maximizing use of commercial technology to reduce acquisition, development, and life time support costs. The panel indicated that the reasonably expected state of commercial enterprise will support some C^3I computing needs. However, its use will require aggressive study of the boundary between COTS, parallel machines, and existing C^3I systems.

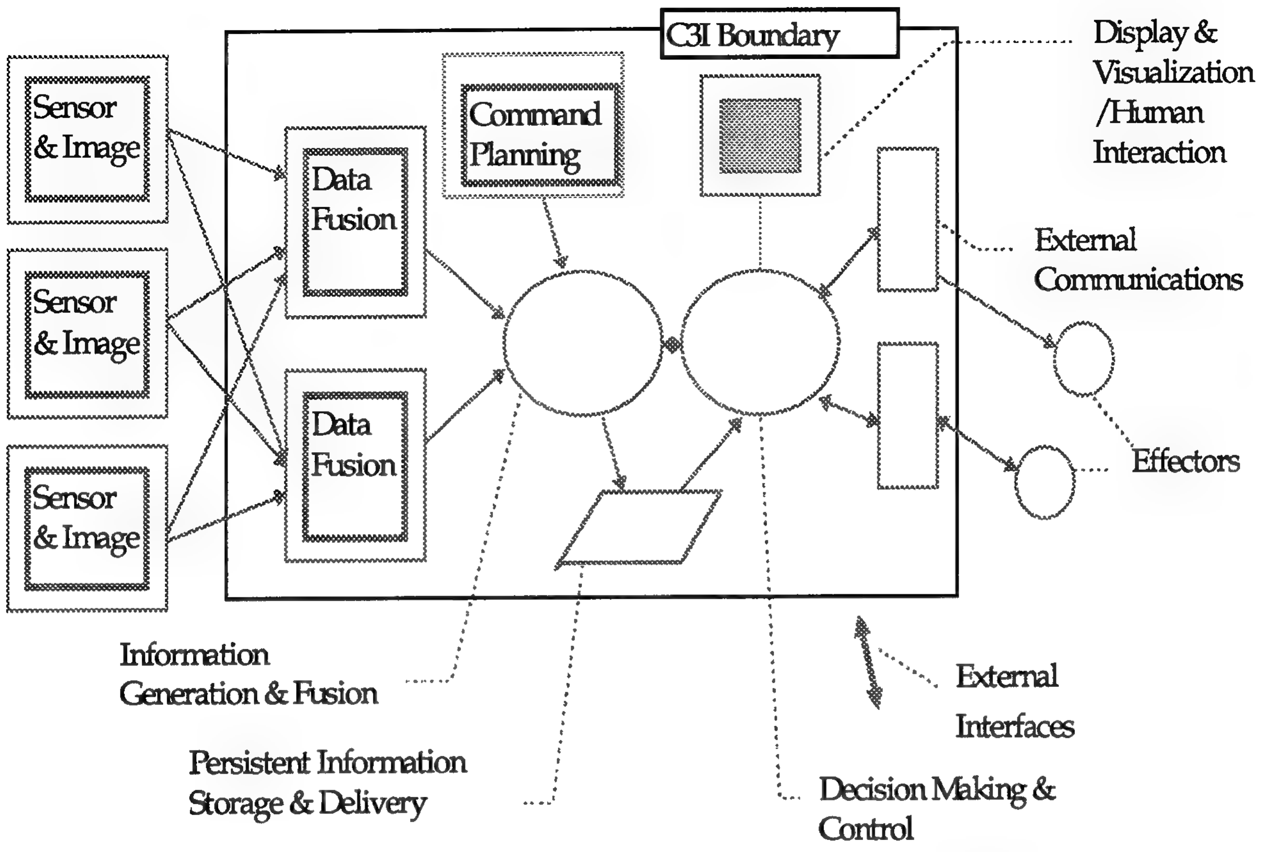


Figure 4-1 Limits of C^{3}I system activities for evaluation of parallel software engineering

4.4.2.1 Rising expectations from COTS technology

The performance and wide spread availability of COTS workstations, networks, operating systems, data and object bases, languages, visualization, virtual reality, voice and pen interaction, and CD-ROM storage are reasonably expected to raise expectations for C^{3}I systems. The panel expects that parallel computing for business, engineering, and other government processes will include many clusters consisting of networked computing nodes with a few four-processor servers. The panel believes that larger tightly coupled multiprocessors will have a market viable only for high-time-value applications.

4.4.2.2 Potential threat from COTS technology

One factor brought out by the panel is that there is the potential for third world countries to use COTS equipment to significantly increase their own command and control capability. Such a threat requires that our C^{3}I builders understand the capabilities of COTS equipment and its potential for command and control applications. Our challenge is to remain far ahead of the COTS capability through innovation and skillful application of parallel computers within a COTS-based C^{3}I fabric.

4.4.3 Parallel software engineering boundaries

The directions of parallel software engineering are not set in a vacuum. Other research technology developments may provide different supporting contributions for C^{3}I systems. Therefore, the panel attempted to observe duplications of parallel software engineering to avoid overlap with other research programs. These boundaries are difficult to establish. Other government programs are in process for research and development areas such as the following: operating systems, object design, scheduling, application programming interfaces, heterogeneous programming, application scheduling, scientific collaboration, distributed system application, etc. Likewise commercial development continues for distributed objects, new operating systems, visualization, multimedia, virtual reality, data base access, protocols for Asynchronous Transfer Mode (ATM), and languages for more effective development of data and object manipulation. The panel believes that its plan should avoid duplication of these efforts.

However, parallel software engineering researchers must establish a relationship with other programs and C^3I providers. Historically, other programs have concentrated on highly structured scientific research and transaction applications that are not effective as a C^3I parallel software engineering technology. An interface to expose the C^3I software engineering needs to other major research efforts is, therefore, necessary. The panel recommended that information on C^3I system needs be communicated to those research communities.

5. TRENDS -- C^3I Mission and commercial-off-the-shelf technology

This section has two major trend discussion sections:

- C^3I demands, technology and mission changes
- commercial off the shelf (COTS) technology

Each of these sections consists of summaries of the position maker's ideas, the panel's conclusions, and an analysis that gives supporting information on the panel's conclusions. (See Annex B for a more detailed summary of the positions.)

5.1 C^3I demands on parallel software engineering technology

5.1.1 Summary of C^3I trend position inputs

5.1.1.1 Future of C^3I Systems

Walter Beam reviewed C^3I and its constituent subsystems. These include the following:

- tactical communications /covert operations
- surveillance/radar/side-looking radar/intelligence activities
- target location and identification, radio location
- identification friend, foe or neutral (IFFN)
- force control and assessment
- tactical intelligence and analysis
- electronic warfare and countermeasures
- command, control, communications countermeasures and counter-countermeasures
- electronic and communication intelligence and analysis

Air Force locations vary significantly in global, aircraft type, positioning, and mobility. The many changes caused by the end of the Soviet Union demonstrate the need for highly flexible systems. The current factors influencing C^3I systems are: downsizing, local conflicts, terrorists, interoperability, wide variety in enemy systems (including US made), grossly altered strategic situation, limitations of technical intelligence, national priorities and overseas basing. The critical military issues that are important to parallel computers in C^3I applications are:

- availability and use of timely high resolution imagery
- rapid, accurate identification of IFFN air, sea and land combatants
- shortening cycle for target detection, identification, attack, damage assessment and recovery
- cooperative operations with Allies and other US Forces
- netting and fusion of target data from a variety of sources and broad-scale intelligence data

5.1.1.2 C^{3}I changing roles and solutions

According to panel members Beam and Wasilausky, future C^{3}I demands for new processing capability will be driven by processing, storage, and retrieval information, data and imagery, advanced multifunction displays and speech processing for human interaction with the system, tactical intelligence imagery, automatic information netting, detection, fusion, data mining, and information warfare protection and disruption of enemy's information. Digital information histories and libraries, multifunction displays and speech narratives will drive human interaction. Both point out that C^{3}I systems depend upon timely information and that future modes of use will be more unpredictable than in prior C^{3}I systems. *High time value in dynamic and changing roles are one of the architectural challenges in parallel software engineering.* In addition, DOD procurement directions will enforce maximum use of COTS technology in C^{3}I systems. Their vision leads toward use of rapid application development techniques from industry that are coupled with continuous exercise of a C^{3}I system. Building system capability through constant interaction leads to evolutionary and adaptive development, allowing the US to maintain its lead in spite of the possibility of the same COTS use by enemy organizations for information warfare.

5.1.2 Panel's assessment of C^{3}I requirements and demands

C^{3}I systems need to use parallel processing to meet response time, physical constraints, and new mission functions. Because parallel computing improves response time, new mission capabilities are feasible. Designers use this approach to radically change mission requirements. They can radically improve a C^{3}I system's war fighting capability by application of parallel technology.

5.1.2.1 Global interfaces and information interchange

The panel expressed the need for a more closely coupled parallel tool set with actual C^{3}I system operations. Understanding the application character, the costs and methods of delivering C^{3}I mission factors, and the necessity for time guarantees requires information collection and knowledge building that many research parallel tool developers presently lack. The parallel software engineering program needs ways of transferring information on fundamentals about the character, factors and high-time-value of C^{3}I systems. Software engineering for parallel systems needs to be in-the-loop to enable its progress to bring the full value to C^{3}I systems. These demands on C^{3}I systems change the importance of interfaces to other systems, typically causing an increased input/output rate.

Parallel computers play an important role as information and data storage and buffering mechanisms for input and output and for transfer of information between C^{3}I system components. Additionally, the parallel computer may transform the organization of data so that each C^{3}I component capability can be maximized toward meeting its time goals. Parallel computers make it feasible for software researchers to create the capability to meet C^{3}I factors, or for the timely execution of applications with a tough C^{3}I mission character. Advanced methods of decision making incorporate intelligent and adaptive methods. They must constantly improve themselves by rule changes or by training. To accomplish this improvement it is necessary that the system have feedback or information gathering capability to achieve its full potential. In-the-loop system probes and data collection are important components for the development of future C^{3}I systems.

The panel projected that an information-gathering probe would be a valuable tool, especially if it were a part of the development process during architectural concept, design, testing, exercises, and life cycle support. Much of the value is the feedback of information on the adaptation and evolution of the system to achieve improved results over time. C^{3}I systems operate within an environment where existing equipment already operates, and where external system interfaces play an important role. Just as business systems have the necessity to deal with legacy systems, so do C^{3}I system components. Due to the long lifetime of

C^3I systems, designers must consider that each parallel component will become a legacy of its own. The model is that parallel systems go into the fabric of existing and future C^3I systems only where necessary. The panel reasonably expects this for high-time-value and tough application character needs. Therefore, the panel expects that an important component will be the interface mechanisms between parallel and COTS systems. System interfaces need a balance of bandwidth capacity and low latency to match the interfaces for high-time-value cases. *Scientific research center methods will not be significant aids in solving this issue because they are generally throughput rather than high-time-value oriented.*

5.1.2.2 Complex data manipulation

The panel expects that present data fusion techniques will increase in complexity. Additional data sources, higher precision, and lower false detection rates are anticipated. In addition, future systems require information fusion. This combines data fusion results into multiple information sets for use in different applications.

These future uses may require complicated data manipulation to such a degree that systems will require parallel computers to meet time goals. *Single workstation technology advances alone are not adequate to meet these needs*. Such processing requirements are, however, of the toughest C^3I character and not generally found in COTS parallel computers. The multiple sets of applications that comprise a command and control system take widely divergent forms. Each form may have widely varying optimum data organizations. Relational data base managers are most prevalent in commercial systems. Their use in accessing multiple data arrangements has not been found to be effective and is difficult to apply for highly diverse component access methods. An inefficiency in both access time and storage volume results. The C^3I system has highly divergent applications with distributed data repositories that do not match a single relational model.

5.1.2.3 Shorter command time scales

The panel agreed that parallel computers most reasonably fit into high-time-value C^3I components. One example is the reduction in time for generation of the command and planning for air battle. The contribution of parallel performance could enable on-line plan changes based on intelligence that arrives during the day. Such an application would require collaborative action in setting up the daily plan and an intelligent systems for safely changing parts of the daily mission. Designers must base the changes on real-time data and information gained during the day's operations.

Time is the essence of C^3I systems. Predictable time response is the essence of C^3I system design. The surrounding sensor signal and image processing must deliver its information without losing information. Buffers provide the means to maintain a constant rate of operation. The result is a steady stream of data pouring into the C^3I interface. This interface must convert the steady stream into detected events. The time to process each suspected event may also vary widely. The event detection component of the C^3I system is highly parallel because each data input stream is a separate channel.

Parallel computing is one method where the C^3I systems can receive the steady stream of input, process event detections, make command and control decisions and meet a time response goal. The decision support actions take several streams of events from different sensor streams. It "fuses" them into a higher level set of event information and associates that set with other information sources. A higher level of fusion might combine these with information based on other data types, such as observations, satellite imagery or human input. The system evaluates these information events, often using empirical methods, to identify objects or activity, determine their value or threat and decide upon the proper response or attack. Detection of possible events and determination of a target must have high signal to noise and low false detection performance. In many cases, the system must perform this processing within a very short time frame, otherwise it is useless.

Therefore, the panel found that time response guarantees for high-time-value applications are necessary for the success of the C^3I mission.

5.1.2.4 Dynamic reconfiguration

The panel predicted that parallel computers in C^3I systems will be required to have the capability for dynamic reconfiguration. In this case, the components that are required in different time domains use common resources. For example, a detected threat or mission change command could redirect processing. In dynamic reconfiguration, the system determines which of the components to run at any given time. By using the resources to best advantage, designers can have the maximum capability for the minimum hardware costs. Often computer resource demands come from new events or from computed information. This requires that the system respond and use its machinery differently. This dynamic change capability is not typical in today's common parallel architectures. Often an event identification perturbs processing control. Also humans can interact unexpectedly, basing their decisions on displays and visual processing. When this happens, the mission may change. The system must switch all its resources to the highest priority and may require that the computation change significantly to perform a new function.

Parallel computation to accomplish these fixed-time response computations must have the same capability. Dynamic resource allocation is a necessity for meeting C^3I mission needs. Of growing importance in university computing is the capability for multiple researchers to interact with a computation as it progresses to reach a better conclusion in shorter time. Collaborative methods will eventually contribute to concurrent engineering or other industrial design and information decision making. This technology might impact a designer's expectations of a C^3I system. Collaborative interaction will be an important factor in increasing the demands for more rapid refinement of decisions through multiple scenarios. Parallel computers will be required for high-time-value application processing and information generation and distribution in collaborative processes.

5.1.2.5 Visual and imagery impact

The panel expects that the C^3I system of the future will move away from simple displays of mixed alphanumerics and plots. C^3I system requirements will follow commercial systems and provide significantly higher information content screens. These include scene images, high resolution vectors, symbolic icons, tables, charts, etc. Providing this extra information may reasonably require video storage, added bandwidth communication, and selection mechanisms for human interaction. In addition, designers will include new forms of image analysis and comparison, possibly directed by human controls. This added capability demonstrates significantly increased expectations of access to information and requires high performance processing. There will be a higher demand on the underlying infrastructure for faster computing results. For example, protocol processing rates may be higher than the workstation processor itself can perform. This is especially true if multiple data streams service the visual processing necessary to meet expectations generated by the capability of the human to ask for more processing and access to data. Visual displays, with much higher information and data content, will drive designer's expectations in future C^3I systems. This may change the organization of processing, transfer and storage of symbolic and real imagery significantly. Both parallel computers and high performance networks may be necessary to perform scene analysis, compression, storage, transmission, etc. Three-dimensional scenes, those possibly provided in new displays like virtual reality, require ray tracing. C^3I applications will have similar imagery for radar scenes, human observations, events, and tracking. The system designer will have to provide image processing and symbolic decision processing in collaboration with information fusion and human interaction. This is a type of image processing that falls within the C^3I system, for information generation and intelligent use for decision making involvement with the user.

5.1.3 Support for panel's C^3I requirement conclusions

The panel's conclusions present a new role for the C^3I software engineer. It demands a higher degree of importance of information dispersion and decision making, increased interaction with C^3I systems, better interoperability, and increased use of COTS tools. The following analysis uses other sources that confirm the panel's conclusions.

5.1.3.1 The warfighter is the informed customer

The C^3I system is undergoing fundamental changes from its present orientation of Joint-Chief's centralized, large scale, response to the Soviet threat into a commander-in-chief (CINC) one. The new goal is to a higher degree of interoperability with the four services operating as a team. The term global infrasphere has been coined to represent the capability provided to the CINC for

> *"... fused, real-time, true picture of the battlespace ... [to allow CINC to] order, respond and coordinate vertically and horizontally to the degree necessary to prosecute the mission ... and be safe from denial, deception, and destruction..." [Edwards]*

The Global Command and Control System (GCCS) is an example of a future C^3I requirement. It has the following core functions:

- crisis planning
- force deployment
- force status
- logistics
- air operations
- fire support
- intelligence
- personnel
- position
- narrative (command collaboration)

[Based on presentation by Lt. General Albert J. Edwards, USAF, Director DISA, "Delivery of the Power of Information Technology to the Warfighter," Defense Studies Board, Summer, 1994]

5.1.4 DOD directions and initiatives

Mr. Anthony M. Valletta has reported [Future Directions and Initiatives in DOD Software Acquisition] that the warfighter should be considered as the informed customer by system developers. The direction of the DOD C^3I software procurement is to recognize that information is critical to the warfighter and will have an expanded role in the future.

> *"[Software] ... needs to be rapidly transitioned to [an] engineering discipline [by] adopting commercial best practice ... [this will] dramatically improve [DOD's] management and software acquisition process ..."*

A major change in software acquisition is that systems are to undergo extensive operational use through use of synthetic environments for training, testing and system assessment against computer generated foes. This combined training and testing is to be made a continuous and daily operation.

5.1.5 C^3I acquisition issues

The Defense Sciences Board, Summer Study 1994, (DSB) expanded on some of these issues. Their view of what the tactical commander requires to be an informed customer by an information system is summarized in the following:

- provide timely information to achieve decisive advantage with total awareness of friendly and enemy situations to gain dominance of all levels of battlespace
- give rapid and reliable movement of information-related active combat needs
- deliver to decision maker and weapon holder with response to CINC/JTF commander and below and tailored to warrior at each level
- create effective but not restrictive security with confidence of protection and graceful degradation

  make information a major discriminator and force multiple through information warfare to deny or disrupt the enemy's information and to accelerate conflict resolution in one's favor

They believed that an expanded role of information capability is to provide the following:

- connectivity of CINC, Joint Task Force, and component commanders
- connectivity among mobile tactical units
- network management and control
- collaborative planning
- interactive video
- distributed database information transfer

The DSB had the same use of COTS technology premise as the panel for software acquisition. That premise is that the government should focus on the military-unique components of C^3I systems and use COTS for information tasks for which it is suited. The need to include information warfare was stressed because information systems are highly vulnerable. They recognize the threat of enemy use of commercially available technology and recommend that software must have incremental upgrade (the forecast panel called this generation scaling) to stay ahead of COTS use in other's hands. They also recognized the need for dynamic reconfiguration to accommodate options in the battlefield for changing mission and responding to threats.

Their results indicated that integrated situation awareness, support to the shooter and continuous analysis and training were to play a critical role in future success of C^3I systems. By integrated situation awareness they declare the need for expanded battlespace picture, imagery, intelligence (SIGINT, HUMINT, and MASINT), timely weather information, digital terrain maps, and other support information. They stress that the information must be delivered to meet specific system needs of the shooter and that real-time is essential to their demands. Planning, training and constant rehearsal were stressed as important to refine the software for these purposes. The problems of information distribution to meet the global infrasphere are the drivers of the need for daily operations and evaluation. They foresaw virtual combat every day as being necessary for meeting the following needs:

- readiness
- acquisition
- debugging
- interoperability verification
- training
- rehearsal
- confidence building
- mission planning
- refining battle damage assessment methods

The DSB recommendation for adopting rapid commercial information technology evolution matches those of the forecast panel. Their software research and development strategy is based on their belief that technology is not the major impediment to information dominance on the battlefield. They believe that the commercial industry leads in information technology and research and that it is globally available. They recommend that the DOD should invest only in research and development of military-unique information technology and use best commercial technology in other cases. This use means more than buying commercial products and services -- it means adopting their practices. Important recommendations are to insist on generation scaling and to reform and align software processes with the life cycle.

5.1.6 DOD procurement practices

The Electronic Engineering Times, Nov. 28, 1994, reported that the DOD has a new strategy for acquisition of weapon-systems. In its article, "The Pentagon begins to shop smart," it gives the following new approach:

> *"[The DOD] ... rests its strategy on ... acquisition reform, streamlining weapon-system purchases, technology investment, and dual-use technologies. ... Federal mandates [are] in place to use commercial off-the-shelf (COTS) integrated circuits and components, the defense industry is scrambling to adapt existing civilian products to military applications." [EE Times]*

The panel, and the recommendations of the Defense Science Board, believes that software should be given the same treatment. Defense contractors are looking at software productivity through the Electronic Industry Association software panel. For example, Motorola's Government and Systems Technology Group's Rose Gibson...chief software engineer and chairwoman of the EIA software panel said:

> *"... [The EIA is] looking for ways to streamline software development by keeping the value-added activities and eliminating the non-value-added," [EE Times]*

The EE Times report goes on with:

> *"... EIA panel ... [to] make greater use of standard architecture while exploiting commercial technologies and other common support and applications tools."*

EE Times also reported that the Defense Science Board has:

> *"... recommended consolidating widely scattered and often redundant software programs ... adopt more commercial practices to control soaring software development costs. ..... product-line development and domain-specific software reuse for leading-edge programs ..."*

Another related statement was given by John Foreman of ARPA who recommended

> *"... adopting a life-cycle approach [to software development] which focuses on the commonality and variability inherent in a family of similar systems. ... command, control, communications and intelligence ... . Arpa has awarded Andersen Consulting, I-Kinetics, and Template Software ... TRP awards for reusable software components [to improve productivity]." [EE Times]*

According to Anita Jones, the Pentagon's strategy is to use:

> *"... Advanced Concept Technology Demonstrations... [to] ... test the Pentagon s ability to integrate defense requirements with commercial production." [EE Times]*

Defense Secretary Perry has:

> *"... directed that performance-based specifications ... be used with the result that ... If a commercial specification is available ... the onus is back on the DOD program manager [to justify non-commercial use]." [EE Times]*

However, some industry representatives recognize that too much enthusiasm for new ideas can lead to unanticipated problems; for example, Gerry Barksdale of Hughes Aircraft was reported as warning:

> *"... that commercial technology wouldn't help except in areas of user interface tools and Ada run-time systems." [EE Times]*

The new desirable software approach is demonstrated in Table 5-A.

TABLE 5-A -- Comparison of software development process paradigms

Traditional C^3I projects	New-paradigm C^3I projects
• Software development process 100% Handcoded Ada development	• Software development process 40% Auto-generated Ada Code 50% Reusable components 10% Handcoded Ada
• COTS software and hardware Select hardware first	• COTS software and hardware Just-in-time hardware selection

[From US Air Force Space Command & ARPA, as reported in EE Times]

The result of this analysis is that the directions advocated by the panel are well founded on directions voiced at the top level of DOD and industry.

5.2 TRENDS -- Commercial off the shelf technology

The second major trend that has significant impact on parallel software engineering is the rapid changes in commercial off the shelf technology. This section discusses some of these impacts and gives a summary of related position maker statements.

5.2.1 COTS trend position summaries

These positions are related to the industry concept level.

- Jon Webb provided the term "high-time-value" as a way of describing the difference between applications that must complete within a given time and those that are to demonstrate speedup. He related high-time-value to closely coupled multiprocessors that are designed to minimize latency. When time does not have high value the concern is the bandwidth necessary to move messages.

- Present industry directions (as well as those latent in announced but not yet outside beta site testing) allow Gordon Bell to posit a future dominant system for commercial information systems. That dominance would drive C^3I systems to maximize use of the same technology for economic reasons. The future system, called Scalar Network & Platform (SNAP), would have a standard desktop client node connected to servers that were at most four processor multiprocessors. The relational database language, Structured Query Language (SQL), would be used to access data from the network. By projecting technology (each critical capacity has been growing by a factor of four every third year). Gordon Bell predicted that by the year 2000, the node processor, memory size, disk capacity, and wide area network bandwidth would all increase in capacity by a factor of ten. Local area network bandwidth capacity would increase, perhaps by only a factor of four. (Wide area and local area bandwidth capacity would converge to the same value.) Both data and multimedia would move across the network, effectively making a large high performance computer from commodity parts.

- In the pure client/server viewpoint projected by Bell there are no parallel programming issues since all analysis is confined to the client. The database manager is the only parallel program and it is used only for data access. A third party vendor would supply it.

A review of this position finds that it is identical to some major software tool vendors, eg, Microsoft, where visual programming is used in this manner for client computing [Source: *Microsoft Developer Network* CD-ROM]. However, commercial publications are already pointing out the necessity for complex and multidimensional data formulations and the need for high-time-value computation. Although seen as simple and straightforward, these expectations are falling short in simple client/server designs [*Client/Server World* and *ComputerWorld*, January 1995]. While many agree with Bell because the largest software vendors are heading toward the SNAPS software version, the publications are identifying a significant need for additional software and hardware capability in application servers.

- Data mining application appears to be one of high importance. George Lindamood pointed out that widespread information access will lead to advanced methods of human interaction to retrieve information. Those methods would likely spin into C^3I applications, increasing the value to the command but also increasing demands on processing capability. Peter Seigel pointed out that globally scalable architectures are necessary for applications scalability for high-time-value applications. These architectures are ideal for data mining (and all high throughput applications) and for high-time-value applications. He believes that algorithms should scale across wide ranging processor counts. Seigel recommends research on lightweight protocols, globally scalable resource managers, object oriented models and high level compilers that link to specialized application needs.

- Kim Gibson pointed out that military systems will require mobile communications services that allow computing resources to be dispersed and flexible. Mobile communications also lends itself to high connectivity. Agile military systems should consider mobile communications as a part of their exercising of C^3I systems and the flexibility required for joint task force operations.

- The competitive forces in the parallel information industry are multi-vendor. The historically massively parallel research vendors may be at a disadvantage because commercial parallel, mainframe and workstation vendors are placing products in the market. Jeff Mohr pointed out that the capability to bring applications from small to large in a smooth transition is important, eg, IBM has an advantage because they can provide MVS and CICS transition from their mainframes. Mohr pointed out that independent software vendors are finding it difficult to move even to small shared memory multiprocessors.

5.2.2 Blue Ribbon Panel's conclusions on COTS industry trends

The panel found that parallel technology will adopt commodity level COTS technology to a greater extent than presently found. The panel believed that the general directions of the SNAP or Network of Nodes was reasonable, but that diversity within that model would require pockets of tightly coupled parallel computers in high-time-value applications

5.2.2.1 Network of high performance personal-computer nodes

The panel's broad vision of the COTS parallel industry of the future agreed with Bell's vision. He calls it the "Scalable Networks and Platforms" (SNAP) architecture. He expects capability to continue to increase by a factor of four every three years. The panel agreed with this version of COTS capability growth and its impact on commercial information systems. SNAP has the following components:

- nodes based on the highest performance but largest volume microprocessor (by the year 2000 this is likely to be an Intel-based computer with 1 GMIPS, 128 MB memory and 20 GB disk drive capacities)
- a four-processor, symmetric multiprocessor would be found at some nodes (the Compaq ProLiant is a present day example, but in the future it will be priced at commodity levels.)
- a single, standard, commodity-priced operating system with network and multi-user capability (Windows NT is the contender --Unix would become outmoded due to high support costs of its many variants)
- Asynchronous Transfer Mode (ATM) would be the local, wide area, and global network connection
- all data access would be by SQL queries into the network or disk unit
- multiple applications could operate at each node
- computing would be peer-to-peer instead of client/server (the fabric of nodes and ATM switches would be a virtual peer-to-peer server)
- no shared memory is provided by this model -- to give scalability of the nodes and network
- metrics of the SNAP network are message delay (latency and overhead) and bandwidth capacity limit delay (gap)

Economy of scale and better learning curves are the primary economic driving factors -- commodity pricing lets almost all applications use the ubiquitous SNAP network of nodes.

The SNAP architectural model did not include "coordination languages" to run concurrent systems over the network, all applications were to be single client-based. Coordination languages let designers set up pipelines or concurrent replicas in a SNAP-like network. Such "middleware" is anticipated by the panel. The panel also reasonably expects that today's message passing, bandwidth capacity dependent applications could fit well on the SNAP architecture.

5.2.2.2 Pockets of tightly coupled nodes

The panel agreed that the general market trend would be toward a commodity market represented by the SNAP network. However, it believes that there remains a commercial market for a range of decision support machines for high-time-value applications. In addition, there will be a growing number of applications that require higher performance on large complex applications. This means that the SNAP network would provide the fabric for future systems but that pockets of specialization would be necessary. The reason for this continued viability is that cost does not drive all computer decisions -- most often the return on investment value does. Organizations often find significant competitive advantage in the high-time-value operations. Many organizations will continue to seek solutions that give them competitive advantage. Also, hardware vendors, with a base of commodity sales to the SNAP-like COTS requirements, would find their highest profit from these pockets of high-time-value. The reasonable expectation is also that the architectural obsolescence in specialized throughput-oriented parallel computers will continue, while the fabric of the C^3I systems structured functions would converge to a stable architecture.

5.2.2.3 COTS network of nodes -- a fabric for C^3I systems?

The reasonably expected vision of a C^3I system incorporates the maximum amount of COTS elements. However, they must also build additions for meeting the high-time-value cases, C^3I mission factors, and C^3I application character. The future challenge is to accomplish advances on the toughest of applications needed in C^3I systems. Designers would use pockets of tightly coupled parallel computers for meeting high-time-value, C^3I factors, and tough application character missions. The panel stressed that a constant exposure at the interface between the projected SNAP fabric and multiprocessors is necessary to understand their boundary. This mixed COTS fabric with pockets of high-time-value machines is the reasonably anticipated network design. It should be the starting point for military and C^3I applications.

5.2.2.4 Anticipated shortfall of COTS capabilities for C^3I

At some point the goals and requirements of the COTS-SNAP architecture and those for C^3I systems diverge. Present examples of COTS parallel computers are too specialized for their specific markets: scientific research, multi-user replicated engineering, fault-tolerant transaction processing, and batch decision query applications. Projected designs for multimedia, communications switches, and replicated engineering also are specialized for particular markets. C^3I applications require very general purpose parallel computing because of unexpected results of data fusion, information fusion, decision making, control, planning -- all in a high-time-value constraint. Most COTS examples are optimized for multi-user throughput. In addition, there is a difference in the depth of concern for integrity factors between military and commercial needs. Some of these integrity factors are the following:

- reliability and fault tolerance
- rapid software development with easy maintenance and enhancements
- security
- graceful degradation
- dynamic reallocation
- physical constraints
- data and network access

A C^3I system must organize and communicate C^3I mission data and information for rapid access by multiple applications. In addition, the time allowed to access information is much shorter than in commercial systems.

5.2.3 Technology trend analysis

Rapid technology advances in computing capability are another trend that impacts C^3I parallel system engineering. The analysis below confirms the pace presented to the panel. The panel used these assumptions while making its strategic plans recommendations.

5.2.3.1 Computer capability trends

5.2.3.1.1 Processor trends

A continual industry rejuvenation is driven by competitive forces and market sizes. These forces require that hardware component vendors stay on a constant path of improvement. This improvement has historically followed a quadrupling every third year progression. Memory sizes available have matched this pace, while memory prices per package remain constant. For example, a single package with 64 KB in 1986-1988 would have over 1 MB today at about the same cost ($50). Microprocessor speed has kept pace with memory size, quadrupling every third year. Secondary storage has also improved its density at this rate of 55 to 60% each year. An additional effect is that the high volume of PC machines has led to quality improvement in manufacturing costs of microprocessors, memory, disks, components and the computers themselves. The instructions-per-second-per-$ delivered by PC technology has almost equaled the pace, improving from about 100 I/s/$ in 1985 to about 1600 I/s/$ in 1992. The most important result is that the cost of software applications has been driven lower due to the large market volume.

5.2.3.1.2 Network trends

Network technology has also seen gains due to open systems and standards. Local area networks (LANs) have increased in performance from 1000 to 100,000 kilo-bits-per-second (kbps) between 1980 and 1992. The latest wide area networks are nearing the same rates, growing from 50 to 50,000 kbps in the same period. Asynchronous Transfer Mode (ATM) networks being demonstrated provide wide area rates at 155,000 kbps, above the 100,000 kbps rates of LANs. The ATM standard has the capability to deliver voice and video at rates and quality necessary for multimedia transmission.

5.2.3.1.3 Professional development workstation trends

If the technology growth curve were followed, today's (1995) typical professional workstation would grow by the steps given in Table 5-B by the year 2000:

TABLE 5-B Capacity Growth

Capability	Size 1995	Size 2000
Memory	16	to 256 MB
Speed	100	to 400 to 1000 MIPS
Removable. storage CD	600	to 2,000 Mbits
Disks	2 GB	to 20 GB
POTs	28.8	to 144 kbps ¼
LAN switch	100	to 155¼622 (ATM) maps
Operating system	Unix	to Windows NT
Languages	C++	to objects
User Interface	Windows	to voice/audio/stylus
Paradigm	Client/server	to agents/mobile (peer-to-peer)
Capabilities	Mail	to multimedia (virtual reality)
Data server	2-4	to 4-16 symmetric multiprocessors
Server language	SQL	to distributed object broker or SQL3 (SQL3 incorporates some Object based methods)

5.2.3.1.4 Semiconductor technology trend

The microprocessor industry is a remarkable one. The pace of change means that limitations of today are soon overcome by time. The prime drivers, microprocessors and memory chips, are driven by semiconductor process technology. The pace is a hundred-fold in a decade. Other key peripherals are doing well at keeping pace -- a 1 GB drive costs less than a 10 MB drive of a decade ago [Slater, *Microprocessor Report*, p 3, 1/23/95]. New applications are anticipated:

> *"Modems have increased only from 1200 bits per second to 28,800 ... a mere factor of 24. When true digital communications become common, ... [they] will increase at an even more rapid pace than CPU power, enabling a vast range of new applications." [Slater]*
>
> *"Today's personal computers ... represent a very early stage in the development of computing. It would be folly to think that the desktop form factor, or the keyboard interface, or any of dozens of other characteristics will continue to dominate."* [Slater]

The Pentium is anticipated to grow to dominate PC systems in 1996 (30 million out of 65 million units) and the "P6" and Pentium to combine at over 80 million units in 1998 [Computer Intelligence Infocorp]. The integer performance of x86 processors has grown by a factor of 200 times in 15 years and is improving at a rate of 50% per year. Compiler enhancements have provided the remaining 5-10% each year to enable the overall 60% growth. RISC processor speed has grown at a rate of 55% per year. The x86 to highest speed RISC gap is a factor of 2 and is increasing slowly. Projections for the year 2000 are for leading RISC processors to deliver 3,000 SPECint92 and for 1,200 in the leading x86 chip. Most of the gains (40% per year) are expected from semiconductor technology, but architectural innovations are necessary to reach the 50-55% growth necessary to reach those figures. [Gwennap -- Microprocessor Report]:

> *"If those techniques do not pan out, radical approaches that break binary compatibility ... may be needed to maintain (or surpass) the 50-55% growth rate."*

5.2.4 Market and technology trend coupling

The impact on parallel software engineering technology is amplified by the coupling of market and technology trends. This section reviews the synergism between the two.

5.2.4.1 Acceptance of parallel processing in the marketplace

Parallel computing also requires broad acceptance of both processor usage and parallel architecture. Some successful forms are emerging in the marketplace. C^3I parallel systems must leverage off those successes and extend them for their military use.

5.2.4.1.1 Industrial parallel software engineering

Success of a common programming model and commercially successful architectures that execute it would serve the C^3I builder well. The trend in production technical computing is toward object-oriented storage for design and analysis information. That trend turns the computing attention center away from pure Fortran applications, and toward use of application packages and system integration languages like C and C++. Most commercial research and engineering production already use applications from independent software vendors. The software that is written is for integration of the packages to make the engineer's job easier. Industrial production centers use that same approach, mirroring the purchase of software vendor and integrator packages.

5.2.4.1.2 Dynamic computer market changes

Today's dynamic changes in the computer market result from the displacement of Emitter Coupled Logic (ECL) processors by Complementary Metal Oxide Semiconductor (CMOS) Logic microprocessors. The result is a rapid advancement in speed of the single microprocessor chips. Low cost and open systems standards make this technology useful as a high throughput (many independent operations per second) computer for workstations and parallel computers. For structured applications with fixed data locations few processors (10 to 30) reach a single processor's (ECL) operational speeds with lower hardware cost. Advocates think that cost-scalable multicomputers built from very high performance parallel CMOS computers will far exceed the capabilities of today's highest performance ECL computers. They also hope that they will be available at a reduced cost per unit of performance.

Large volume production of microprocessors is necessary to recover development and foundry costs and to allow a margin for continued development. The most popular models will rapidly evolve and improve while the less popular ones become obsolete and disappear from the market. The Intel i860 is an example of volume inadequacy and architectural performance limits, causing obsolescence. While the Intel 80x86 / Pentium series is a successful volume case. A processor applications base and choices of operating systems are critical factors for market success in personal computers and workstations that use microprocessors.

In parallel systems the microprocessor cost is not the critical issue -- many highest performance microprocessors are available for $500 per package, while multicomputer nodes are priced at 100 times that cost. Even the design and foundry costs of a specialized microprocessor are not overwhelming if foundries are available. The critical factors are the development cost of a robust system, compiler software, operating systems and the existence of an application's base. Achieving that base requires the investment of many independent software application developers. Broad market acceptance of a microprocessor is necessary for attracting the investment in operating system, languages and applications. For example, IBM, Motorola, and Apple combined forces for ensuring a broader market, adequate volume and applications base for the PowerPC microprocessor. HP and Intel jointly announced an advanced processor to replace the Pentium series.

These changes offer new opportunities for enhancing command, control, communications and intelligence systems. Market-coupled, commercial-off-the-shelf (COTS) solutions allow C^3I builders to meet their budgets. Application specific parallel processors necessary for meeting tight physical constraints could build hardware to execute the same common model, gaining efficiencies in software development.

Because of the recognition of their effectiveness, commercial information systems routinely use high performance parallel computing for on-line transaction processing and decision query servers over a range of sizes and architectures. Database and transaction applications for parallel computers have been ported and are available from several independent software vendors. These products span both symmetric multiprocessor and local memory based multiprocessor architectures. An example is Sybase DBMS on the AT&T Global Information System product line with both SMP and distributed memory architectures. The success in this field show that if parallel machines are adequate and useful for an application, independent software vendors will respond. C^3I systems could use this type of processor for applications that match them. (See Table 5-C.) The suitable applications are multi-user transactions, database storage, and replicated copies of programs, one copy per user. AT&T's robust interconnect structure in its model 3600 has proven reliability. More important, the operating systems for the commercial system are robust and reliable.

The server might be a four-processor one that provides files for several users. Much of this parallel processing is now limited to replicated applications for different parameter sets. Unfortunately, a single large application, such as simulation or detailed operations of an engineering process, gets little speedup gain from the present server technology. Often engineers wait overnight for critical runs because the necessary processing performance is not available. If the four processor servers delivered speed on a variety of engineering processes a sixteen-processor system would be justifiable in these environments.

Symmetric multiprocessors provide a few primitive mechanisms (e.g., threads coordinating via locks) to allow the programmer to deal with concurrency. The thread programming model is often not portable with other vendor's hardware and is not portable to local-memory-based multiprocessors. Sales of these machines, database and transaction processing software, and a full spectrum of business process applications is now available on those machines. The popular client (a workstation) and server (a parallel processor) operation allows many corporations to spread their operations across the nation on wide area networks. The commercial software package industry has built a significant inventory of software, making skilled and trained personnel with experience available. C^3I systems often require the similar transaction, data base, and reporting applications. C^3I system designers should understand and capitalize upon these capabilities. Due to the small processor count, these processors cannot solve some of the large-scale problems of C^3I systems. However, they can contribute a significant level to selected matching application types.

These systems are also finding applications in industry for support of decision support systems. All of the well-known examples communicate with an operating legacy system. The systems extract data and feed it into the parallel computers for processing. They work on large and complex queries and analysis applications. They often cut completion time down from weeks to days. However, there is a demand for even quicker results. The primary disadvantage is that each example is ad hoc and requires a significant staff for development and operational support.

Large-scale commercial applications have expectations of mainframe-like operations (general applicability) but at much higher performance. Multitasking (switching between multiple users to overlap and hide delays) makes interconnect latency less important to getting high throughput for transaction and database applications. However, a single user with a large decision support application with complex queries requires multiple processors. AT&T provides software that breaks complex queries into independent parallel transactions for execution on either SMP or capacity-scalable systems. This allows use of local memory based multiprocessors for complex queries. However, large scale parallel systems still do not have the general applicability of mainframes. The vendors of large scale parallel systems have "skimmed the cream" of transactions and data access. (See Table 5-C.) Their products do not provide the versatility of programming and effective operation provided by the supercomputer or mainframe on complex and dynamic applications.

Early examples of commercial systems that have embraced parallel methods show that the present successful parallel offerings cannot replace entire systems. Large information systems have additional capability or function instead of simple replacement of "legacy" or "heritage" or "dusty deck" operational systems. Industry now uses parallel computing to accelerate processing of new throughput-oriented applications. However, the mainframe or supercomputer remains the operational, day-to-day production system for heritage or legacy applications. Often the prior software investment is too large for a parallel system to displace an operational system. The proprietary file systems upon which these systems have been based compound this problem. The investment to shift to a parallel system is significant and requires the following:

- extract data from the legacy system
- insert it into the new parallel system

- do the decision processing
- post analysis
- put some results back into the legacy system (possibly)

Often this effort requires tens and sometimes hundreds of people. Over the next five years both commercial and C^3I systems will likely consist of mixed legacy and advanced technologies. Software engineering progress is necessary to build this mixed operational capability to allow incremental insertion of parallel technology.

One would not expect high effectiveness on clusters due to high protocol latency in protocol-based parallel systems. Possibilities include carrying out a small-packet, message-passing mechanism in software for both symmetric multiprocessors and for the buffer exchange hardware for capacity-scalable designs. Synchronization latency would be similar in both cases: highly portable code results. With a similar latency and common programming model, applications are portable across a range of sizes and architectures.

The parallel computer industry is experiencing significant competition at all levels of systems. In the Intel 486/Pentium marketplace AT&T, Pyramid, Sequent, TriCord, Compaq, and Encore plus many other smaller vendors are already present. Compaq has already sold several hundred thousand multiprocessor systems. Intel recently included key parts of the shared memory consistency control within the Pentium microprocessor chip. In addition, Intel has announced a standard for Pentium systems that will allow the commodity "clone builders" to join the symmetric multiprocessing hardware market. Cyrix has announced a competing open standard to prevent Intel patents from cornering the market.

5.2.4.2 Information system impact

The following technologies could play an important role in different parts of the C^3I system:

- Information systems -- desktops, servers, decision support, data access -- business
- Engineering -- simulation, design, software development -- product development
- Embedded -- sensor, imaging , physically constrained missions -- medical, process control
- High performance -- scientific research, grand challenges -- weather, large scale aerospace, safety and environmental simulation

Table 5-C identifies computer technology sectors that impact C^3I parallel software engineering.

TABLE 5-C -- Future Influence on C^3I System (and vice versa)

	Tech-Impact	Fraction of C^3I	Potential for Change by C^3I Software Engineers
Information system	Large	> 50%	Little -- has own agenda
High performance -- decision support	Yes	Selected points	Significant: information fusion and decision
Engineering	Some	> 10%	Some -- Environments and System Design Tools
Embedded	Some	Selected points	Interface & Simulation; use as high performance decision computing
High performance -- Scientific research	Small	very small	None from scientific research;

Information systems (combining PC and commercial) now have the largest potential impact on computer technology due to their massive market size. Table 5-D shows the fraction of the $112 billion commercial market (1994) that goes to desktops, commercial and technical computers. Some workstation and mid-level computer vendors now target the business application as their market of choice, leaving engineering as a second market. The combined information system market is soon to be driven by PC/workstation, network and server technology, standards and economics [Microsoft].

For the larger portion of its computational resources C^3I system technology can adopt and ride the information technology curve [Wasilausky]. The demand for parallel computers beyond that required in information system technology is a question that depends upon the justification line between military-unique, board-level COTS and COTS system units in an Air Force war fighting information capability. The panel defined as external to C^3I the sensor/ image processing input and response "effectors" that perform actual weapons delivery. Parallel computers for those external systems are now engineered to fit into physically constrained situations.

TABLE 5-D 1994 Computing Market -- $112 billion [Williams]

Market Segment	Fraction	Component	Comment
PC	50%	<10% technical	90% commercial
Commercial	35%	~75% transaction	small, but growing UNIX transaction processing
Technical	14%	~50% workstation	mid and mainframe is greater than supercomputer and parallel

Board-level COTS technology also plays a major role in building sensor and image systems [Lund]. Historically, those systems have used signal processing instead of commodity microprocessors and real-time instead of UNIX workstation operating systems. High performance computing technology, as defined by the scientific research and grand challenge machines, was seen as having little impact on the choice of technology curve for the bulk of C^3I computing. However, high performance parallel machines for information fusion and decision support that are matched to C^3I needs were believed to be possible areas of importance.

5.2.4.3 COTS business system trends

Business information systems are undergoing a radical change in organization as a result of reorganization and flattening of management, a process often called process reengineering. The change is closely coupled with creating distributed processing operations called "client/server." In the client/server organization a client (desktop workstation) uses a network to access data located on a server. The most cost-effective form is for the server to be a Unix-based, highly robust symmetric multiprocessor which is dedicated to the data base operation. These servers now cost significantly more than home computers of equivalent processor speed and memory due to their robustness, fault tolerance and interface to the Unix operating system. They cost significantly less than mainframes often used for the same purposes.

The interface to the operating system has created a barrier to commodity pricing of up to four processor units. Standard interfaces to the operating system have been proposed by Intel and a consortium of its competitors. That standard will allow commodity builders to enter the market. Thus, a standard data access language has emerged from the commercial marketplace (SQL) that allows client tasks to obtain data safely in a distributed network. Software tools for building user interfaces based on access to SQL databases are now readily available, eg, PowerBuilder or Gupta. Emerging are also object storage systems and mixed SQL and object systems, eg, UniSQL, Illustra and Object Design. Object systems based on the common object request broker and other distributed object mechanisms are also on the market.

Operating systems are also in flux. (See table 5-E.) The need is for a standard Applications Programmer Interface (API) that allows applications on one workstation to be ported by recompiling to run on others. The Unix market is suffering from vendor inability to define and support a single standard that accomplishes a common API [Bell]. Although the common open systems environment (COSE) builds an aggregation of function calls (1170 of them) from past dialects, the fragmentation of the operating system leaves each platform vendor with increased support costs. Those support costs are a burden to workstation vendors that are reflected in increased costs and loss of competitiveness. The largest vendors keep extensions and unique file systems that lock in their customers, creating the potential for dramatic switching to new applications. A new common standard is likely to appear from the commercial PC-based market, Windows NT. (Windows NT will run on all major platforms.) This allows server platforms for applications that are not tied to particular variants of UNIX, but to a high volume industry standard [Bell]. Cross system environments provide this function (Galaxy, Powersoft, NextStep, and Taligent's planned product). Any standard must also avoid proprietary SQL calls.

TABLE 5-E Database Application Server Plans -- Operating System

[*ComputerWorld* - 1/23/95]

	1994	1996 (plan)	1996 (potential)
Unix	46%	46%	16%
Windows NT	10%	20%	34%
Netware or OS/2	10%	6%	6%
Unsure	34%	28%	44%

This result (based on interviews of 50 IS managers by Forrester Research) shows that there is a high chance of drastic change in the database application server operating system of choice.

5.2.4.4 System technology trends

The technology trends allow a designer to devise a system that consists of computers with professional workstation capabilities, connected via ATM-based switches. These would use the most common standards as the basis for client applications. This organization can match the throughput of large parallel computers by the same methods used to specialize the present parallel computers -- having multiple users on each node to hide latency, using multiple paths to access data by distributing storage in redundant disks in the network. Parallelism would be limited to data-base operations. Clients applications would use transactions to access data and perform processing. Scalable data-bases would provide implicit parallelism. Scalability is assured because no shared memory coherency mechanisms are used.

Evolvability (higher performance commodity technology can be inserted as it arrives) allows the design to constantly ride the commodity technology curve. One ingredient is the constancy of the SQL access. Some small symmetric multiprocessors would be used because they, too, meet the common multi-user operating system and SQL-based operations. Almost all programming would be client programming since the SQL servers would use purchased standard systems [Bell].

There are two compelling reasons for the emergence of this information system vision of the future:

- massive standardization gives massive use
- economic forces are enormous.

Both are a result of the driving force in information systems:

> "*...the user investment is in programs, data and training...*" [Bell]

5.2.4.4.1 Scalable network and platform architecture

The model of multiple PC nodes, ATM network, and SQL data access was called Scalable Network and Platform (SNAP). The term SNAP-like is used here as a generic name, representing several similar organizations with a range of capability. The working session of the panel believed that, in general, the bulk of commercial information systems would head in the direction of SNAP. This assumes that several barriers are overcome, eg, legacy interfaces, limitation of servers to be commodity multiprocessors, latency (~12 microseconds) of over 10,000 instructions for ATM access, limitations of SQL for high-time-value applications, lost speed limitations of ATM for data transfer (the network may be tuned to make voice and video guarantees instead of data reliability guarantees). All these concerns, excepting the legacy interface are due to a need for high-time-value, mixed character, and getting mission-like factors, which are sometimes required in commercial systems.

The resulting marketplace projection if this vision is fulfilled is that PC and Network machines would totally dominate, leaving only some very high cost applications that justify workstation or mainframe legacy needs. (See Table 5-F.) The pure SNAP vision leaves out high-time-value applications in the commercial world that cannot fit on a single PC (projected to be a 1,000+ MIPS machine) that gets its data by SQL (and multimedia) calls at 155 to 622 mbps.

TABLE 5-F Year 2000 Computing Market -- $250B (The SNAP Vision)

Market Segment	Fraction	Component	Comment
Set top	--		Included as part of SNAP
Single PC	5%	Standalone use	Single, non-interfaced PCs
PC -- SNAP Connected	85%	Transactions - all types	All Commercial and Multimedia Systems; SQL and Multimedia
Other	5%	~0% transaction	~ mainframe legacy market only
Technical	5%	~100% workstation	~ no supercomputer and parallel

5.2.4.4.2 Trend to recognize need for business logic processing

On-line analytical processing (OLAP) is a new phenomenon in client/server computing. Its need has arisen because the form of computing being practiced is that of using a single workstation desktop as a client (requester) to data servers (replicaters and responders). The single client does the graphical user interface to interact with the user to generate requests for data. In this "Visual-to-SQL programming model, all business logic analytical processing is performed by the desktop and all data are obtained via SQL calls to servers. *The SQL access has been found to lack the capability to effectively access complex and multidimensional structures found necessary in some applications.* Relational databases are fine-tuned to volume "throughput" processing, making them hard to access outside the defined relational structure. They also have poor performance on complex data access operations. According to one decision support user:

> *"You can't easily build in intelligence or dynamic analysis [into the relational structure]." [ComputerWorld]*

OLAP technologies allow the user a "spreadsheet" metaphor to access data more easily. They use different storage methods to allow faster access. However, they maintain the model that all servers provide data and the client does the processing on it -- leaving no capability to go beyond the capability of the single workstation. (Multiprocessor workstations would overcome this limitation.) The tools to easily move a user from single desktop to multiprocessor desktop may come later when the limits become more apparent [ComputerWorld, p12, February 20, 1995].

5.2.4.4.3 Specialized Processors for On-time and Integrity Factor Processing

The panel anticipates a need for high-time-value application servers and for meeting C^3I mission factors. It expects that similar high-time-value application needs will arise from corporate competition sources. Therefore, the panel expects that an alternative design will emerge to fulfill those needs.

The panel's trend projection is a SNAP-like one with differences necessary to provide high-time-value resources where they are needed. It also would have tools for system engineering and programmability that let designers contemplate a design that is client only. Therefore, the forecast panel believes that implicit object oriented methods would be provided to client programmers to allow ease of programming any high-time-value applications on suitable machines. Market forecasters apparently believe in the value of object-oriented methods, eg, see Table 5-G. In addition, the architecture would provide interfaces to legacy systems. This new architecture seems a reasonable projection since an object-oriented standard could arise to compete with SQL, because of its reusable code advantage. Also, a legacy interface problem solution is necessary for acceptance into the business community. Capital budgets are not adequate to replace legacy systems in a single step. In addition, some expect that there will arise more concern for high-time-value applications when the commoditization process brings many firms into the same level of system capability. Firms that experiment with faster decision processes will find that their complexity increases the demand for high-time-value processing significantly.

TABLE 5-G Market forecast for object-oriented platforms [*ComputerWorld*, 63 - 1/30/95]

Platform	1994	1998
Visual Development Tools	$777M	$3.45B
Object database management	$233M	$1.39B
Object-oriented operating environments	$40M	$1.22B

6. VISION -- Parallel Software Engineering

6.1 Positions on parallel software engineering

This section reviews the positions on software engineering presented to the panel. They form a sense of the state of the art and vision of research leaders in the field. (Some positions are based only on the paper provided.)

6.1.1 System level positions

- The report by Masimi Uemoto described the viewpoint of engineers that are seeking tools to do distributed or parallel engineering. Uemoto sought solutions that fit into the normal engineering network environment, but she found no effective solutions in spite of reviewing a number of reported demonstrations of parallel engineering tools. Parallel and distributed tools are not effectively entering the engineering process although the necessary technology has been demonstrated for a decade.
- Yalamanchili described a demonstration of collaboration and interactive execution for real time and scientific applications that are needed for complex data sets whose path cannot be foreseen. Their approach fits into the projected needs of C^3I systems to respond to mission changes.
- J.C. Browne added that there is a growth in complexity of systems that will prevent software implementation that is efficient and portable. He advocates integrating parallel structuring with traditional function and modularity, executable abstract specification, and similar engineering (simulation and verification at each abstraction layer) as done in hardware systems.
- Armand ten Dam advocated a support environment for structured design tools targeted to real-time applications.
- Jay Eckard provided the view that large data sets are controllable using commercial database tools and that new technologies, eg, solid state disks, can change data access performance dramatically.
- Carl Murphy proposed that C^3I factors and tough application types are those places where parallel technology is needed. He believes that object-oriented methods can be used to solve the C^3I factors and that tough applications require hardware and operating system kernel changes. Murphy also pointed out that evaluation of parallel computers on cost of peak performance instead of cost of life cycle has led to specialized parallel machines that exacerbate the software engineering problem. He is a believer that there are solutions to building general purpose parallel machines for the tough decision problems in C^3I systems and that they would be successful on the commercial market.
- Murphy, Seigel and Wasilausky all believed that dealing with specialized needs, such as C^3I factors and tough applications, requires object oriented methods that operate at the same level as the operating system kernel.

6.1.2 Programming model positions

These positions cover virtual machines and automated parallel programming.

- C^3I system and application engineering need architecture-independent software because of long life cycles and mission change. Refinement layers representing design-through-code are current research solutions. Douglas Smith and Jan Prins described refinement approaches that start with an executable specification and refinement layers for portability. In this scheme mission changes would be identified again at the specification level and trickle down to the hardware code level. Hardware obsolescence would only effect the branch of the refinement tree for the architecture type. In this manner automated program generation support is provided and changes are made within the tree only for the effected components. Prins provides performance assessment and Smith adds automated program generation to the method.

- Stephen Yau described a layered design approach based on objects. His tools get portability by targeting a common intermediate layer for code generation on different machines. He claims advantages in conceptual parallelism identification, ease of modification, and reduced life cycle costs and effort.
- Tom Cheatham described the Bulk Synchronous Parallel (BSP) model that represents the same concept level for parallel machines as does the von Neumann model for sequential. In this model programmers can estimate the performance of an algorithm without coding and trying it. BSP requires a super step barrier at which time all communication and synchronization of the step must be complete before the next step is started. Cheatham described the compiler, library, optimization and correctness proofs and other refinements for general purpose programs. BSP has potential impact on C^3I systems because its model can be matched to fixed time frame methods used in real time coding. A theory that combined monotonic rate analysis and BSP would be the foundation for a useful tool in parallel high-time-value applications. The BSP model of superstep barriers also is well suited for distributed processing applications that are projected for the COTS-SNAP distributed system.
- Rajive Bagrodia described the requirements for a C^3I Virtual Machine (CVM) as a design process and programming model. CVM incorporates multiple computational paradigms (task, data, vector), heterogeneity, real time and reliability issues. Performance estimation would be included at the design level.

6.1.3 Programming tool positions

These positions covered approaches to programming tools.

- Andrew Sherman described the virtual shared object model (VSOM) to improve on shared memory and message passing. Their concept is based on a content-addressable memory with which each process sends self-synchronizing messages. This allows each process to get information by name, not location. VSOM allows dynamic and complex system integration. Commercial tool examples are Linda/Paradise and database management products.
- Craig Lund provided information on the lack of suitability of scientific parallel computers for C^3I systems. His comment was --

  > *"Scientific programmers usually think of their problems in "control flow" terms ... In contrast, real-time signal and image processing applications constantly receive new data. Therefore, programmers of such systems usually think of their problems in 'dataflow' terms ..."*

- Lund described a component programming proposal for signal, image, and other sensor processing needs. He also noted a culture difference between defense vendors and academic institutions and commercial firms. *The challenge of product fund raising requires product firms to understand how their products fit into common business or engineering processes and the market. Researchers have no need to follow through to the marketplace.*
- David Rich stressed that C^3I systems will consist of some of the largest and most complex systems to be built and that COTS parallel development tools may not be adequate for such applications. Commercial language and tool vendors are targeted toward smaller projects. This is confirmed by the

observation that the present COTS industry direction is confining user interfaces to single client station programs done with corresponding "visual" tools. Systems are assemblies of open system parts including the database application. His debugging tool product experience indicates that a programmer needs to be able to debug within a single development context, requiring that the debugger operate on all components in the system.

6.1.4 System effectiveness tool positions

These positions covered the means of increasing effectiveness (speed related to processor capability) of parallel programs.

- Martin Davis described a method of creating a hardware architecture after the application is designed.

  "No matter how hard one tries to isolate the application programmer and user from the hardware, the hardware is the ultimate determinant in the performance of a program."

- This work has a role in C^3I systems that must be physically constrained or meet special environmental factors.
- Min-You Wu provided the following comment:

  "While regular applications can be solved efficiently, only a small fraction of applications are regular. ... the current technique cannot solve applications with irregular ... structures ..."

- Wu described run-time systems for implementing complex applications.
- Sanjay Bhansali and CS Raghavendra believe that problems of increasing scale and complexity are the problems of importance. They point to the need for effective distributed applications in future high performance networks. They believe that domain-specific components will be the important path for software engineering. They have sequential code reverse engineering tools that allow automatic mapping to parallel computers.
- Robert Rabb gave a specification and design language for large grain applications that is intended to give portability, design correctness and reliability. Run-time communication and synchronization are defined at design time to obtain these results.
- Matt Rosing described flexible and efficient abstractions for parallel programs based on compile time language constructs.

6.2 C^3I system development methods

The necessity to use system and software engineering methods to build C^3I systems is due to their size, high-time-value, application character and mission factors. Since C^3I design and development requires many people, the visibility of the design and its implementation are very important. *Parallel computers exacerbate the situation because their performance (time guarantee) varies widely when the design mismatches the architecture.* Designers allocate function and performance to components when designing a C^3I system. This is difficult even when the underlying capabilities of the computing resources are consistent and steady providers of computing capacity. Without the capability to predict the behavior of a parallel element in the system the design risk becomes excessive. The panel believes that designers should not use parallel computers unless they identify clear advantages, ie, they should not use technology for its own sake. However, it also believes that the system designer should aggressively design for applications that require performance levels only obtainable through parallel computers.

To conceptualize and develop an architecture and a design that delivers high-time-value guarantees requires knowledge of dependable and predictable completion times for each component. In addition, the extra computation required to deliver all required C^3I mission factors must be predictable at design time, prior to detailed mapping steps to an architecture. The panel found that delivering reliable time of completion predictions on diverse application character is a necessary design tool. Designers also need tools to help in delivering C^3I mission factors and predicting the associated performance penalty. Tools that provide information about the application character are also necessary. It is also necessary to develop methods to incorporate C^3I factors such as time guarantees, resiliency and integrity, and physical constraints imposed by operating in aircraft.

6.3 Status of parallel technology and software engineering

The assessment of the panel is that present COTS parallel computers have not provided an effective capability for delivering C^3I mission factors such as reliability, availability, security, fault tolerance, and graceful degradation. The panel expects that C^3I mission factors will continue to be important differences between C^3I and COTS parallel systems. However, software engineering tools that accomplish C^3I factors within COTS-based systems would have high value in commercial systems.

The application characters of C^3I systems are very different from scientific research, business transaction processing and other well-known parallel applications. Those character differences are a combination of complexity, interaction, synchrony rate, and resource demand. Extensive (unstructured), symbolic (logical and decision) interaction, frequent synchrony, and dynamic resource demands are the most demanding characteristics. The panel referred to these as the "tough" applications.

In contrast, "standard" applications typically have a structured, (large) message passing, infrequent synchrony and static resource character. The greatest challenge to applying parallel computers to C^3I systems is in dealing with tough applications for high-time-value applications. The panel viewpoint taken is that C^3I systems should have hardware components of commodity COTS, specialized (possibly parallel) COTS, and militarized components. These components would be workstations, networks, specialized processors, operating systems and run-time software for implementing C^3I factors and meeting tough character applications.

A future view of C^3I systems is one based on a high level of available COTS computing machinery and software. The panel raised some concerns about over-enthusiastic COTS adoption and the possibility of a lack of consideration for long-range disadvantages of COTS. One example given was market obsolescence. The panel reasonably expects that significant cost advantages will result. *The panel recommends setting the goals of parallel software engineering by reflecting on the future of COTS components.*

The panel recommended a strategy to maximize the use of COTS, but use parallel computers to meet high-time-value goals, C^3I mission factors and application character. This is to be accomplished with total cost conscientiousness, not hardware cost alone. This strategy requires programmability combined with performance and portability. It also requires that the underlying system concept have a close relationship to COTS where it applies. An important requirement is to know how to determine the relationship between COTS-SNAP and parallel computers. That interface is expected to be one of the challenges of a parallel software engineering research program.

6.3.1 C^3I software engineering

An important tool that is necessary to design the interfaces in C^3I systems is a suitable computer aided system design tool. *These tools are necessary to allocate function and performance to components, to predict their performance and to estimate* C^3I *mission factor capability.* A designer typically iterates their design until they meet all mission needs. One result of the panel's vision is the identification of a tool to determine the interface between the SNAP-like architecture COTS and the parallel machine part of the system. As the designer attempts to allocate functions the design tool would predict performance on the SNAP-like COTS architecture. When the designer cannot meet high value time limits, the tools should recommend a series of sizes of parallel architectures to meet the high-time-value goal. Thus, the tool lets the designer find the edge between the SNAP-like COTS fabric and parallel or specialized components.

The system design tool provides for an overall system allocation of components. In addition, programmers need application development tools for design and programming of applications. The position makers described several methods of application engineering that included performance prediction and specification execution. Position makers presented refinement methods, object-oriented methods, specialized components and virtual computing models to the panel. The panel believed that the goals of these methods were appropriate. *The panel recommended that a mechanism be set up to put these methods more intimately in the loop of* C^{3}I *exercises, testbeds, and applications.*

The panel found that discussion of architectures and applications were facilitated by reference to the Bulk Synchronous Parallel (BSP) model of barriers with defined terms for communication-to-computation ratio and barrier delay costs.

The panel thinks that tools are useful that let C^{3}I system engineers emulate SNAP-like architectures on present-day parallel computers. Developers would have to normalize for projected bandwidth capacities, processing, and storage capability. The BSP model could be useful in that normalization. If engineering tools could match the architecture to a latency cost model, it could be easier to develop virtual machines as targets for programming tools. Experiments in emulation of predicted COTS systems provide a test bed to allow quantitative performance engineering that attains C^{3}I mission factors. In addition, designers could explore techniques for testing of tough character applications.

6.3.1.1 Object-oriented design, development, and programming

The panel reported that object-oriented design and development methods have made a significant impact on display screen programming, data exchange between independent applications, and organization of access to data objects. Commercial system and application programmers have accepted these methods. Small programming groups are making significant productivity gains through object reuse, easier software maintenance and efficient enhancements. Panelists raised questions about how to overcome performance loss in C++ when compared to compiler optimized C, how to best use C++ in large projects, and how to provide better design visibility.

Object-oriented methods are playing an increasing role in commercial systems. This technology plays a key role in creation of displays and has reduced the screen building and human interaction programming task significantly. Likewise the languages used by commercial support staff is also becoming object-oriented. System design techniques are also following this path.

Some of the panel members believed that SQL was not adequate for the commercial systems of the future. They found that programmers would most likely program using a combination of visual object-oriented languages, such as Smalltalk, and business logic languages like C++. As a result, the Blue Ribbon panel's concerns over the effectiveness of SQL for complex and diverse type C^{3}I applications are confirmed by current reviews from the commercial industry.

6.4 Analysis of COTS trends in parallel software engineering

6.4.1 Object-oriented technology

Cheryl Ball reported in *Client/Server Today*, February 1995:

> *"[the] first wave of object-based tools ... enabled the first generation of client/server database access applications ... [they] revolutionized information access in many corporations. But the next wave ... will allow rapid development and deployment of flexible enterprise-wide business processing applications."*

Sixty-eight percent of companies surveyed and reported in Balls report say that object-based tools are important to their redesign efforts.

That object-oriented methods have solved many graphical user interface (GUI) development needs was demonstrated at the Database and Client/server World Conference and reported in *Client/Server Today*, February 1995. The example given is MCI's Friends&Family marketing system. The system was put together in three months using NeXT's object technology. This rapid development for meeting agile marketing needs is expected to increase in the future. That meeting confirmed the idea that SQL may not be adequate for complex and diverse data source C^3I systems. One reviewer gave the following analysis of database management systems based on object-oriented and SQL methods:

> *"Object Oriented methods have good support for the following: complex and user defined data types, error recognition and correct handling by DBMS and mapping of object class to domain data type. On the other hand, SQL is excellent for simple transactions but is not a good or true implementation of relational technology, has no concept of domains or data types, and does not work well as a repository for objects because the object class to relations is not as effective as direct object implementation."*

6.4.1.1 Object-oriented effectiveness

Object-oriented programming effectiveness was discussed in *ComputerWorld*, p 114, January 30, 1995 where a Smalltalk object programming example was reported. Table 6-A gives the results.

TABLE 6-A Productivity improvement with Smalltalk object programming

	Original task	with Smalltalk
Development time	19 months	3.5 months
Labor (months)	152 person months	10.4
Lines of code	265,000 lines of PL/1	22,000 lines of Smalltalk

(Original source: International Data Corp. Framingham, MA)

ComputerWorld went on to say that object databases are typically used in technical databases. However, their use is recognized as meritorious when there are the following requirements:

- the application has complex data structures
- there are many relationships between data structures
- the relationship is as important as the data
- high performance is required for data manipulation

- behavior modeling is as important as data modeling

The conclusion from these inputs is that object-oriented programming is confirmed for small projects. The client application programming, however, can be used in large projects where a few top level programmers build the system interfaces and connection and other programmers must have only a client program to develop. Their effort is to be blind to the architecture of the system. This model (and the low relative cost of Unix servers compared to mainframes) has led to the recent success of the "client/server" approach. Parallel software engineers need to adopt the same model, where almost all programmers on the team do architecture-independent programming. A few programmers would do the system programming to distribute the applications into the distributed system and a few others would write any high-time-value programs that execute on the low latency multiprocessors in the system. Adopting this commercial model as a goal of its research could benefit the C^3I parallel software engineering process.

Annex A
Parallel Software Engineering Assessment

Annex Summary

Viewpoint and Issues

This assessment looks at parallel applications and architectures from the viewpoint of a C^3I system builder. There are many purposes for parallel computers in C^3I systems. Therefore, understanding applications is the starting issue. C^3I systems have physical and integrity constraints beyond the design intent of commercially available machines and operating software. Therefore, architectures will continue to be a critical issue. The power and dynamism of the computer technology marketplace significantly influence both applications and architectures. The market is a third critical issue. Software engineering is to provide the processes, tools, and metrics that allow engineers to design and build complex C^3I systems. These processes and tools must deliver the fast development cycles, limited budgets and low life cycle costs now expected of peace-time military development.

Applications

The contribution of parallel computing is in reducing the time required to complete an execution, thus advancing the capability of C^3I system applications.

C^3I applications have fundamentally different time response goals. A discourse about applications without an understanding of their time response goal leads to a cacophony of different interests. This assessment stresses that the time response goal is the critical factor in discussing the lines of tension between applications, architectures, market forces and software engineers. These lines of tension are:

- Design-Time Matching of Application Needs and Constraints to Architectural Capability
- Tools and Metrics Applied to Applications and Machine Performance
- Programmability and Machine Portability Models

Our assessment includes an architecturally independent rating of C^3I parallel applications by time-response type and by characteristic criteria. Together these are a characteristic pattern of execution. A comparison of the pattern of today's parallel application successes to C^3I patterns shows that the architectures are ineffective for many C^3I needs. We find that we must make a new focus to broaden the understanding of C^3I applications and the ways that parallel architectures can solve them.

Architectures

Scientific-research parallel computing is not general enough to provide a complete model for C^3I system applications. Client-server data base and on-line transaction processing, a successful example of throughput-oriented parallel computing, are limited to serving many independent users. *These present parallel successes fail to solve mission critical applications, those in which an execution has a specific period to complete its execution.*

Alas, these successes have a significant penalty. Builders experience the inability to design with assurance that the system can execute in a predictable time on a chosen architecture. They experience the penalty when the effort to fit the application into the architecture grows beyond original budget estimates. In addition, they have no structured way of engineering for rapid response computing, real time predictability, system availability, fault tolerance, and security. The C^3I industry needs a robust hardware parallel architecture and software engineering processes that lead to significant performance leverage, while maintaining programmability, portability and maintainability. *We conclude that today's parallel computers fail to deliver their promised performance in complex Air Force C^3I systems because their specialization prevents adequately general software.*

Market Force Strategy

The Commercial-off-the-shelf (COTS) strategy is one in which commercial competition provides advances in capabilities needed for military systems with only marginal investment by the government. Often overlooked is that the savings in software costs due to mainstream market acceptance is the most important factor in COTS strategies. *When designers overspecialize parallel architectures to reduce hardware costs, they often lose the benefits of COTS provided software.* Operating systems, programming tools and languages, and data base systems are fall-outs of COTS successes. However, if these cannot meet the application demands of C^3I systems, then the COTS strategy fails. The gains of a COTS strategy are an illusion if the result is not adequate for general acceptance.

Software Engineering Issues

Software engineering is a discipline that deals with the large, complex, dynamic, mixed hardware of C^3I and large scale systems over a long life. Parallel processing technology now exacerbates the development process by reducing portability, increasing software costs and creating performance uncertainties. Advocates of machines specialized for scientific research often recommend them for applications for which they are inherently unsuitable. The necessary changes in hardware, operating system and support tools make the conversion unrealistic and risky. The result is an industry state often described as "lack of software." Parallel computer technology must adapt to attain the performance benefits without process uncertainty.

The software engineer needs a set of tools that measure the capability and generality of parallel computers. The development process should be based on an architecturally independent model and architectures evaluated on how well they execute that model. Once the engineer knows the characteristics of the application's execution and its patterns of operation, they can predict a machine's performance for the application. Such separation of hardware and software development is important to the engineering of parallel hardware and parallel software. Without this capability, each effort by hardware vendors and software developers is ad hoc. The result of a successful parallel software engineering activity is a wider knowledge of C^3I needs and demands. In addition, the recognition that business mission critical, decision and process control systems need similar capabilities would allow vendors to build more general parallel solutions to serve this wide application range.

Annex A: Parallel Software Engineering Assessment

1 Overview

1.1 Scope

This assessment looks at parallel applications and architectures from the viewpoint of a C^{3}I system builder. Figure 1 - 1 shows these issues. There are many purposes for parallel computers in C^{3}I systems. Therefore, understanding applications is the starting issue. Budgetary limits point toward use of commercially available parallel computers, yet C^{3}I systems have physical and integrity constraints beyond the design intent of commercially available machines and operating software. Therefore, architectures will continue to be a critical issue. The power and dynamism of the computer technology marketplace significantly influence both applications and architectures. The market is a third critical issue. Finally, we must bring all these together coherently. Software engineering is to provide the processes, tools, and metrics that allow engineers to design and build complex C^{3}I systems. These processes and tools must deliver the fast development cycles, limited budgets and low life cycle costs now expected of peace-time military development.

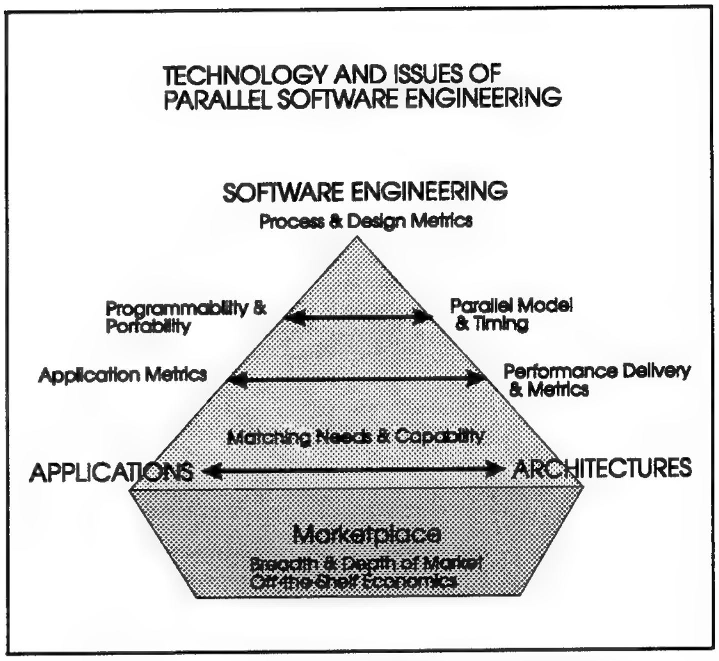

TECHNOLOGY AND ISSUES OF PARALLEL SOFTWARE ENGINEERING
SOFTWARE ENGINEERING
Process & Design Metrics
Programmability & Portability
Parallel Model & Timing
Application Metrics
Performance Delivery & Metrics
Matching Needs & Capability
APPLICATIONS
ARCHITECTURES
Marketplace
Breadth & Depth of Market
Off-the-Shelf Economics

Figure 1 - 1. Parallel Software Engineering

1.2 Applications

The contribution of parallel computing is in improving the time response or thoroughness of the result, thus advancing the capability of C^{3}I system applications.

C^{3}I applications need parallel compute capabilities. Present systems could better fit their physical constraints (size, weight, volume, power, etc.) with effective parallel computing. Parallel processing at the source of data reduces communication while maintaining the required data rates. Parallel event and information fusion create the knowledge needed to make decisions faster and more thoroughly. Designers can bring global information to bare on local command decisions and vice versa. Performing multiple simulations and iterations improves battle plans and scenarios. Large simulations can provide models for response to global or environmental changes. A reduction in execution time makes each opportunity possible.

Designers and builders of C^3I systems generally have a specific mission problem with specific physical constraints. To create a design the mission is typically broken into component subsystems. Each component has a period during which its execution must complete to act in concert with the other components. Parallel computing offers both a reduced response time and improved capabilities because more computation can complete within the system's time constraints.

However, each of these opportunities has a fundamentally different time response goal for its acceleration. A discourse about applications without an understanding of their time response goal leads to a cacophony of different interests. This assessment stresses that the time response goal of an application is the critical factor in understanding or discussing the lines of tension shown in Figure 1-1. These lines of tension are the problems facing parallel software engineering. They are the following:

- Design-Time Matching of Application Needs and Constraints to Architectural Capability
- Tools and Metrics Applied to Applications and Machine Performance
- Programmability and Application Portability with Effectiveness

Our assessment includes a rating of C^3I and parallel applications by criteria that are independent of the particular parallel architecture. Together these criteria form a character or pattern of execution. By observing successful parallel applications with a similar character or pattern, one might identify architectures that are suitable. A comparison of the characteristic criteria of parallel application successes to C^3I characteristics shows that today's parallel machines are ineffective for many C3I needs. We find that we must make a new focus to broaden the understanding of C^3I applications and the opportunities that parallel architectures can provide in solving them.

Command, Control, Communications and Intelligence (C^3I) systems must respond to multifaceted threats and scenarios. These systems contain a variety of computing types and subsystems. Parallel processing is necessary to improve both throughput and response time demands. Parallel processing needs improvement in availability, real time response, security, fault recovery, and physical parameters. The system software needs of parallel processing includes operating systems, languages, debugging tools and controls consistent with parallel machines. It needs reliable, portable and reusable software. Present programming processes are too costly and difficult. Air Force C^3I application complexity makes it difficult to use the present generation of high performance parallel hardware, system software and applications. A key ingredient in the difficult is the design of predictable rapid response components for complex, unstructured elements of applications.

C^3I systems have some characteristics similar to business and scientific research ones. However, they have a broader set of characteristics than the more narrowly focused applications supported by commercially available parallel computers. This essay proposes a classification scheme. The first typing is by the length and degree of time guarantees for completing an application. The time-of-execution types we propose are the following:

- Continuous Data Rate (maintaining a constant level of data throughput)
- Hard Real Time (guaranteed maximum response time)
- Rapid Response Time (generally statistically meets quick time goal; graceful operation during occasional failure to meet goal)
- Interactive Single Job (visual display of results and rapid response to single user)
- Interactive Multiuser (meets combined delay and throughput benchmark)
- Capacity Constrained (delays of several minutes are allowed to respond to user)
- Capability Constrained (results are not needed for immediate use; delays of hours or days allowed; used to improve quality of system operation)

The last three in the time response classifier list follows Furtney and Taylor's classification for workstations and servers [Furtney and Taylor 94]. This typing shows that a parallel architecture must provide very general speed improvements to have a broad market. Therefore, the C^3I research investment should concentrate on achieving dependable, across-the-board, response-time improvement. It should support only those multiprocessors that can respond rapidly for a wide range of general applications.

Typical C^3I systems consist of many components, each with a response time goal or hard limit. Predictable response time is critical to our interests. Therefore, our focus is on response time reduction for very general, concurrent applications. These span all the time response types. Unfortunately, scientific processing application characterizations apply to the parallel computers that are specialized for structured scientific applications. These are typically limited to Capability Constrained time response types. A business oriented characterization would be limited to Interactive Multiuser (for on line transaction processing) and Single User (for decision support data base access) time response types.

The C^3I classification extends the one proposed by Worlton for scientific computing. Worlton's classification splits each of four criteria into halves and rates applications on their fit to the resulting sixteen criteria. His four criteria for scientific computations are:

- Degree of Parallelism (High or Low Concurrency)
- Parallelism Uniformity (High or Low Uniformity)
- Communication Distance (Local or Distant Messages)
- Grain of Synchronism (High or Low Communication Grain)

Worlton's criteria is too simple for consideration in rating C^3I applications. We propose that the following new set of four:

- Interaction Mechanism (Symbolic or Message Interaction)

- Resource Demand Fluctuation (Static or Dynamic Resource Demand)
- Complexity (Intensive or Extensive Complexity)
- Synchrony Grain (Frequent or Infrequent Synchrony)

1.3 Architectures

Scientific-research parallel computing is not general enough to provide a complete model for C^3I system applications. Client-server data base and on-line transaction processing, a successful example of throughput-oriented parallel computing, are limited to transaction applications tuned to serve many independent users. *These present parallel successes fail to solve mission critical applications, those in which an execution has a specific period to complete its execution. Other failure cases are where a few users need effective use of the entire capability of the parallel computer for solving a complex or dynamic application.*

Gains in throughput processing via parallel processing have, so far, have a significant penalty. Builders experience the inability to design with assurance that the system can execute in a predictable time on a chosen architecture. They experience the penalty when the effort to fit the application into the architecture grows beyond original estimates. The extra effort often compromises the schedule and cost budget. In addition, they have no structured way of engineering for rapid response computing, real time predictability, system availability, fault tolerance, and security. *We conclude that today's parallel computers fail to deliver their promised performance in complex Air Force C^3I systems.*

The C^3I industry needs a mixture of a robust hardware parallel architecture and software engineering processes that will lead to significant performance leverage, while maintaining programmability, portability and maintainability. The system designer must also include physical constraints, time response predictability, fault tolerance and security. Designers have few research results about these topics on today's machines due to the concentration by academe and industry on scientific research.

Due to their more tightly-coupled structure, multiprocessors have the potential to reduce the execution time of concurrent applications. Our classification stresses communications content and latency that match the parallel system to its potential for a dependable reduction in an application's response time. These are:

- networks-of-computers - for selected applications with restricted communications and loose time constraints (Suitable for Capacity and Capability constrainted applications)
- multicomputers (cost-scalable) - for selected applications with limited communication and time goals over a narrow size, structure, and dynamic range
- multiprocessors (capability-scalable) - for more general applications with some communications and time limits, capable over a wider range of application structure, dynamic operation and sizes
- symmetric multiprocessors (count-limited) - for servers in distributed systems with

relaxed time limits and for applications with time constraints that can be met with small processor counts

There are many competing parallel technologies. Client-server systems, clusters and heterogeneous computing methods provide choices for meeting computing needs. The parallel system software engineer must address the allocation of requirements between parallel processors, clusters, networked computers and heterogeneous systems. These choices form a hierarchy of applications that must match the computer architecture's interconnect capacity. Understanding how a language or tool fits into the hierarchy is critical to evaluation of its value.

Industry understands the software research needs of throughput computing. Applications are On-Line-Transaction-Processing (OLTP), Data-Base-Management-Systems (DBMS), and replicated applications. Clusters and networks of computers are effective for high-throughput processing of independent tasks and replicated processes. Heterogeneous computing mixes network and tightly coupled computing. One form even breaks an application into specialized parts to match specialized processors.

Architectural trends are toward a dynamic and multifaceted industry that is on the verge of establishing itself as a major market. Competing forces arise from multiple vendor sources: "scalable" parallel systems, symmetric multiprocessors, commercial on-line-transaction processing, data-base-management-systems, and mainframe vendors. Trends are toward the following:

- a smaller maximum processor count
- higher bandwidth capacity interconnects
- hardware supported programming enhancement mechanisms
- increased capability for input/output, memory, and disk access
- increased availability

The trend is also toward improving programmability. Costs per processor node are tending upward as vendors target an engineering production instead of a research laboratory market.

1.4 Market Force Strategy

There is a constant, cost-driven pressure to move to less costly semiconductor technology. That trend results in reduced demand for large mainframes and supercomputers and an increased one for microprocessor based parallel computers.

The Commercial-off-the-shelf (COTS) strategy is one in which commercial competition provides advances in capabilities needed for military systems with only marginal investment by the government. COTS components have the promise to provide an affordable technology for building high performance C^3I systems. Often overlooked is that the savings in software costs due to mainstream market acceptance is the most important factor in COTS strategies. *When designers* *overspecialize parallel architectures to reduce hardware costs, they often lose the benefits of COTS provided software.* Operating systems, programming tools and languages, and data base systems are fall-outs of COTS successes. However, if these cannot meet the application demands of the C^3I system, then the COTS strategy fails. For example, turning scientific-research-oriented parallel computers into real-time, signal-processing systems may require new operating systems to meet military reliability, availability, security and real time response guarantees. If vendors over-specialize to meet scientific research market needs then the system builder may have to reengineer many architectural capabilities to meet the new demands, eg, delivering constant input/output bandwidth rates. The gains of a COTS strategy are an illusion if COTS systems are not adequately general for myriad application types and characteristics.

The economy of scale of commercial hardware and software leads to wide spread use of COTS workstations. As a result operating systems, data base systems and high level languages are readily available for scientific and commercial uses of workstations and personal computers. The success of workstations and their ever increasing performance and programmability leads to the same high expectations for parallel processing. Commercial parallel processing industry success is important to the use of parallel processing in C^3I systems. C^3I system developers need economy-of-scale hardware and software for parallel system, applications, tools, and development processes. Presently parallel processing attracts too few independent software vendors.

Highly competitive markets drive the technology of computing. As a result, technology significantly affects the application of parallel computing. The parallel machine OLTP and DBMS industry shows the success of large processor count parallel computers in multiuser applications. Business information system builders now embrace small-processor-count symmetric multiprocessors for those applications. (Both Compaq and Sun sell over 10,000 units per month as file and query servers.) Workstation, symmetric multiprocessor and scalable parallel vendors now compete in the same market. This convergence shows that parallel computers have market viability. However, few independent software vendors are aggressively porting functional applications to parallel computers. Until they do, the market may remain viable but constrained.

1.5 Software Engineering Issues

In sequential computing there is a quantitative measurement capability based on instruction timing and the mix of instructions in a typical application. In parallel computers, performance benchmarks and application suites are not adequate.

Software engineering is a discipline that deals with the large, complex, dynamic, mixed hardware of C^3I and large scale systems over a long life. Parallel processing technology now exacerbates the development process by reducing portability, increasing software costs and creating performance uncertainties. Advocates of machines specialized for scientific research often recommend them for applications for which they are inherently unsuitable. The necessary changes in hardware, operating system and support tools make the conversion unrealistic and risky. The result is an industry state often described as "lack of software." Parallel computer technology must adapt to attain the performance benefits without the process uncertainty of parallel processing. Engineers need methods and processes that lead to an accurate representation of both applications and architectures.

The software engineer needs a set of tools that measure the capability and generality of parallel computers. The development process should be based on an architecturally independent model and architectures evaluated on how well they execute that model. Once the engineer knows the characteristics of the application's execution and its patterns of operation, they can predict a machine's performance for the application. Separation of hardware and software development is important to the engineering of parallel hardware and parallel software. Without this capability, each effort by hardware vendors and software developers is ad hoc. This approach has little potential for reaching the large markets necessary for commercial advantage. The result of a successful parallel software engineering activity is a wider knowledge of C^3I needs and demands. In addition, the recognition that business mission critical, decision and process control systems need similar capabilities would allow vendors to build more general parallel solutions to serve this wide application range.

Workstation designers use application benchmarks and quantitative measures (SPECmarks) to guide development of microprocessors and workstations. Parallel computer designers have no similar quantitative foundation to allow performance estimates for different machines. Parallel computers have no accepted set of commonly accepted instructions upon which to base the measures. Researchers continue to devise nonportable mapping schemes that apply only to a restricted type of applications. Amdahl's law continues to apply in spite of anecdotal results of restricting application types to defeat it. As a result, parallel computers have not escaped the trap of low efficiency and unfulfilled promises of peak speeds.

The software engineering process needs a programming model that is architecturally independent and quantitative. For these reasons, the software engineering process lacks an integrated set of parallel tools .

Software engineering for parallel systems suffers significantly from a too rapid obsolescence of hardware, a lack of common programming models, and the requirement to map to obtain reasonable performance. The general process of software engineering has evolved as a basis for managing the building of long-lived, complex C^3I systems. Software engineering processes for conventional sequential computing systems have a built-in factor-of-safety from the progress of technology. The obsolescence of parallel computers, the long procurement cycle, and the lack of an adequate quantitative model for performance assessment of parallel computers negatively affects the software engineering process.

2.1 C^3I System Applications

C^3I systems need to use parallel processing to meet response time, physical constraints, and new mission functions. Because parallel computing can improve the response time, new mission requirements become feasible. Designers use this approach to radically change mission requirements and can drastically improve the system's war fighting capability.

Command, Control, Communications and Intelligence (C^3I) systems are large systems that must respond to multifaceted threats and scenarios. C^3I systems typically deal with complex warfare situations and dynamic scenarios. They respond to multiple forces, to dispersed geographies and unexpected threats across a spectrum of weapon technologies. C^3I functions are both highly complex and highly concurrent. Their components include: transformation of multisource data into information, fusion of information into intelligence, automated and human controlled decision making, simulated scenarios, force and weapon's system responses, communications to higher or lower command levels, etc. Burdens of responsiveness, availability, fault tolerance and recovery, security, and reliability, are severe in C^3I systems. As a result C^3I System components include a variety of subsystems and interactions, some of which are the following:

- Real time - sensor data conditioning, acceptance, and preprocessing
- Data Communications
- Information Base Access
- Information Fusion & Evaluation
- Event Detection Decision
- Scenario Based and Real Time Simulation
- Action Path Decision
- Response System Control

2.2 Assessment of Parallel Applications

2.2.1 Parallel Processing Application Typing and Characterization

The assessment approach is to define a time response goal and a set of application criteria. These define an application's character profile. Determination of an application's character could allow a designer to estimate the effectiveness and feasibility of applying parallel computers to C^3I system components. These components have a widely varying requirement for execution completion time. This varies in length and guarantee of repeatability. In this section, we give a set of response time goals and application characters. We use a response time goal, frequency of synchronization, complexity, resource demand (or path) stability, and interaction mechanism as our criteria. The capability of today's parallel industry for C^3I applications is limited to a narrow range of capabilities. Instead a general parallel capability to meet all C^3I character types. C^3I systems are too complex, too dynamic, too real-time constrained and too reliable to make good use of scientific research computers for all their needs.

2.2.2 Response Time Application Typing

C^{3}I systems include many subsystems. This means that the constituent parts often have a different response-time type. Table 2 - 1 shows a proposed C^{3}I oriented typing.

Table 2 - 1 Application Types

Application Types by Response Time	Characteristic
Capability Constrained - Hours Frame (Uses capability of largest parallel machines)	Long turnaround time allowed, could use the maximum capability of parallel computing available
Capacity Constrained - Minutes Frame (Full Capacity required for up to an hour)	A delay of several minutes or tens of minutes allowed for the computation (need several iterations per day)
Interactive Multiuser *	multiple users operate on each processor, sharing the processor resource gives a high transaction rate
Single User Interactive*	yet a goal is set for interaction with users Single Job Interactive * fast interaction with a single user is necessary
Response Time *	Response Time Goal - Graceful response- to-failure to meet time
Hard Real time *	Guaranteed maximum response time
Continuous Data Rate *	Perform a continuous process at a minimum input data rate; reduce the input rate to a lower output rate; (buffers typically provide continuous rate to allow for catching up when a rigid time goal is missed)

Table 2 - 2 rates various applications.

Table 2 - 2 Some Parallel Applications by Response Time Type

General Category	Examples	Usual Response Time Type
SCIENTIFIC Large Scale Scientific Research Codes - Grand Challenges	Weather Modeling Climate Prediction	Hours to Day Frame (Capability)
TECHNICAL Engineering Applications	Semiconductor modeling, Electronic Design, Mechanical Design Chemical and Molecular Modeling, Fluid Dynamics, Combustion, Biomedical, Composite, Manufacturing Design, Materials	Wanted to be Minutes (Capacity) but most often Hours (Capability)

General Category	Examples	Usual Response Time Type
BUSINESS Client Interaction Applications	Customer Interaction OLTP Data Base Management System	Multiuser Interactive Multiuser Interactive Multiuser Interactive
Business Reaction	Knowledge Based and Neural Network Decision Automation Data Base Management System	Single Job Interactive Response Time Goal
Business Decision Support	Strategic Decision Query Data Mining	Hours to Days Time Frames (Capability)
Communications Applications	Voice Response Switches Multimedia	Constant Data Rate (for Voice) Response Time Goal (Telephone) Multiuser Interactive
COMMAND, CONTROL, COMMUNICATIONS & INTELLIGENCE		
Image Processing & Understanding	LIDAR	Constant Data Flow (Processing) Response Time Goal (Understanding)
Command Decision Support		Single Job Interactive / Response Time Goal
Information Base Access	Shared Situation Data Base	Multiuser Interactive
Data Fusion	Target Tracking	Response Time Goal
Information Fusion & Evaluation	External Events combined with Target Events	Single Job Interactive
Event Detection Decision	Event Matching	Automatic - Guaranteed Response Time Manual - Response Time Goal
Action Path Decision	Command Scenario Matching	Response Time Goal
Planning and Scenario Simulation	Battle Plan Preparation	Single Scenario - Hours Frame (Capability) Multiple Scenario - Minutes Frame (Capacity)
Real Time Simulation	Event Projection & Prediction	Response Time Goal
Process Control	Weapon System Control Aim & Fire Missile System	Guaranteed Response Time (Real Time)
Visualization	Battle Situation Display	Interactive
Real time	- Data Communications & Distributed identification Data Delivery	Guaranteed Response Time (Real Time)
Real Time	- sensor data conditioning, acceptance, and preprocessing	Guaranteed Response Time

2.2.3 Application Character

Worlton and Associates have published a classification of scientific computations based on scale of parallelism, uniformity of communication, distance of communication, and granularity of synchronization. C^3I applications need a more robust character set due to their wide range of mission needs. Figure 2 - 1 gives a sixteen-way division for four criteria that are more suitable to C^3I applications.

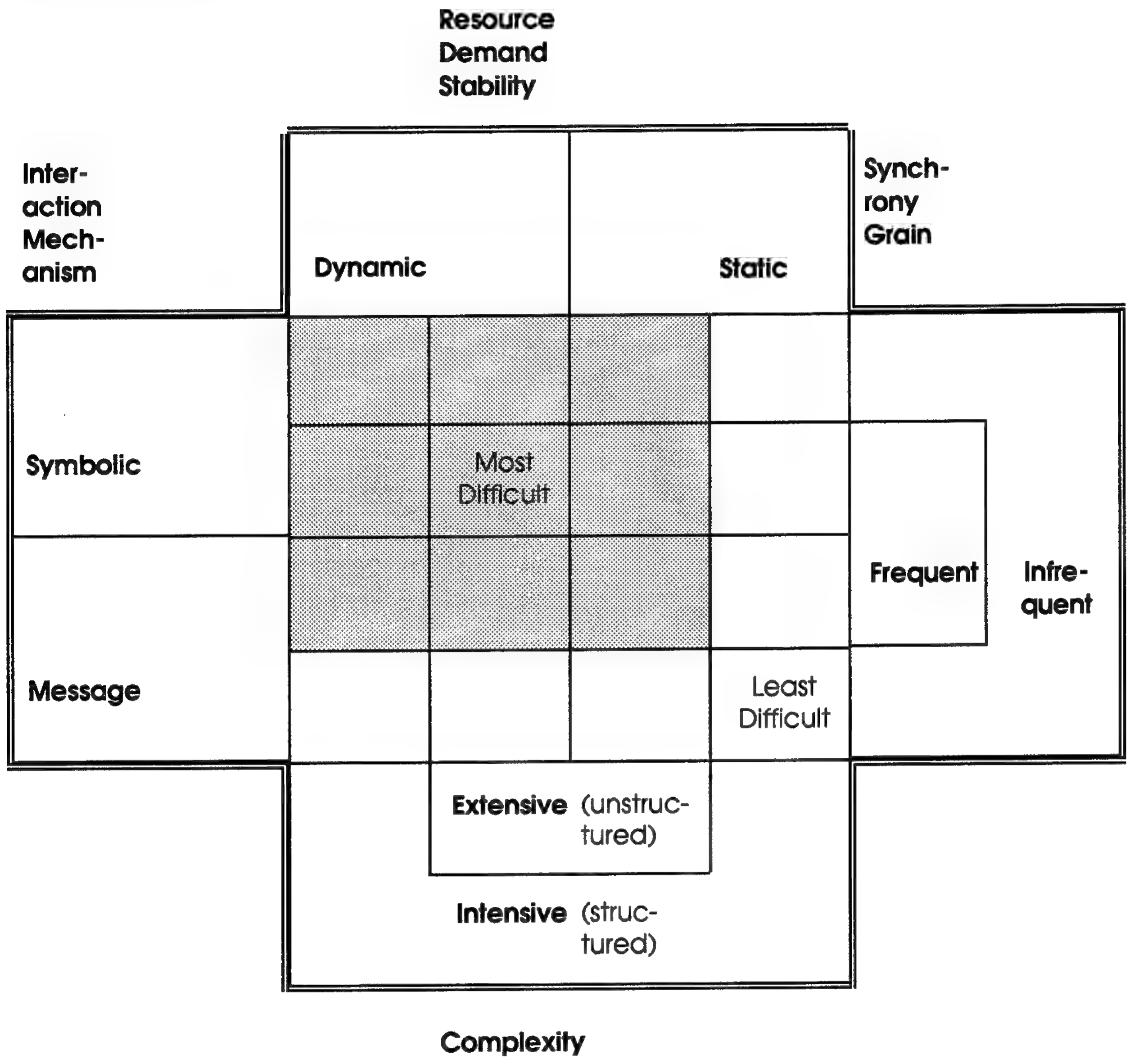


Figure 2 - 1. Characterization of C^3I Applications

- Complexity is extensive if one cannot analyze and understand its paths ahead of execution. One can predict the paths of intensive applications because of a defined structure.

- Interaction is by message passing if all communications consist of movement of data, symbolic if control synchronization is present.

- Path Stability is dynamic if resource demands can change based on the decisions during processing, static if resources are rigidly assigned.

- Synchrony Grain represents the ratio of parallelism operations to ordinary instructions. Frequent implies a higher fraction of parallelism synchronizing operations when compared to ordinary instructions. Infrequent implies a small fraction number.

According to this scheme the most trivial parallel application character is intensive, communicates via large messages, has static resource demands, and infrequent synchrony. Replicated programs operating on fixed, resident data are of this type. The most difficult character is extensive, interacts symbolically, changes resource use unpredictably, and has frequent synchrony. Parallel operating system kernels might be an example of the most difficult type.

2.3 Parallel Application Review

Our character rating should allow an evaluation of successful parallel applications. This should show how well the computers upon which the successes were based might support C^3I applications.

2.3.1 Scientific Research Applications

Baskett and Hennessy [Baskett93] describe some parallel techniques used in scientific applications. They give the following problem types and the techniques for obtaining high performance:

- multipole - treat clusters of particles
- direct matrix - do blocking and tiling
- iterative - combine grids from discrete grids (multigrid)
- spectral - tile

Tiling means grouping of clusters of frequently communicating tasks to reduce communication between processors. These methods are necessary to obtain effective operation on today's parallel computers. Note that applications are often selected to match the machine as a research project. They are not, as is necessary in C^3I applications, constrained according to C^3I mission requirements.

These applications are Capability Constrained time response types. Vendors tune the operating systems, user tools, programming languages, and infrastructure to this response type. The character of scientific and engineering research applications tends toward intensive, message interacting, infrequent synchrony and static resource demanding. Spectral methods may interact symbolically, that is operate in a data presence driven mode. Similar application character are often found in the white zone on the bottom and right side of Figure 2 - 2 where the character visualizer is filled in for several applications.

2.3.2 On-Line-Transaction and Decision Support Queries

Data base queries for on-line-transaction-processing and decision support systems are another successful application of parallel computing. These are the multiuser interaction time-response-type. Their unit of interaction is a transaction. These systems have the character of intensive, frequent synchrony, static resource demanding and message passing. However, some OLTP systems do have extensive portions hosted on each node. The complex interaction is necessary to keep five or more users active on each node and to overlap disk accesses. For example, a sequential program converts complex queries into multiple transactions and parses them out as transactions. Decision support queries may have extensive portions to parcel out transactions to perform queries in parallel.

An analysis of the interconnect structure of large parallel data base machines shows a highly capable interconnect structure. This was necessary for supporting both frequent synchrony messages in the system and the message passing demand. Symmetric multiprocessor machines maintain balance by keeping the processor count low.

2.3.3 Communication and Multimedia Switches

Communication switches are another successful example, although most of those presently in use are special purpose PBXs and not general-purpose parallel machines. They typically do telephonic switching, but vendors plan multimedia applications. These are constant bandwidth time-response-type applications. They are intensive, message passing, with infrequent synchrony. (Synchrony activity is only used for the selection of the switch route.) The switch application resource is static design.

Forecasters predict that the delivery of multimedia services is a future large market application. This new application may reverse the trend toward lower connectivity because when the computer acts as a specialized switching unit, matching channel input to output under infrequent interactive commands, the higher connectivity is necessary. Multimedia switching is a constant-bandwidth time-response-goal application. Its character is infrequent synchrony (subscriber requests), message interacting (long multimedia data streams), static resource demand (always at full capacity) and intensive complexity. It has a high availability and a high input-output requirement.

2.3.4 Signal and Image Processing

Signal and image processing are examples of special purpose parallel machine successes. They are constant-bandwidth time-response-type, although the systems they are found in may have other time response types. One characteristic of these applications is the lack of storage. The system brings in signals, converts, and combines them into events to transmit to the rest of the system. These applications are intensive, message passing, with frequent synchrony with static resource demands. Frequent synchrony results if interaction between fast fourier transform butterfly nodes are distributed. Parallel signal processors of this type are found in many C^3I systems. A variety of real time operating systems and libraries are available. These applications may also used in process control systems but not as the controlling processor. Other application successes are those that can be placed in attached processors. These have a character of intensive, message passing (large blocks), infrequent synchrony, and static resource demand with respect to the host processor.

2.3.5 Replicated and Multiple User Engineering and Business Applications

Many common parallel engineering applications are code and design replications on several processors with one parameter set per processor (Single-program, Multiple-data). These applications are most often Capacity Constrained or Capability Constrained. They are the simplest example of intensive, message passing, static resource demand and infrequent synchrony character. In many cases the parallelism can be created interactively by the engineer using network commands. Interaction is via message passing. The resource demands are static and the synchrony is infrequent. It appears that many applications with this character are suitable for operation on clusters as well as on multicomputers and multiprocessors. Other commonly known applications tend to contain a significant fraction of extensive code combined with a core of intensive solver code. Evaluators often cite NASTRAN as an example As presently formulated, it is not effective in a single-program, multiple-data mode unless different processors perform a copy of NASTRAN on different parameter sets.

2.4 Character Rating of Selected Applications

Figure 2 - 2 gives the approximate character of application types discussed above. Note that no general application can be specifically placed in a character square -- only a specific one. This display shows the place that these applications would tend to be found. That is, along the message passing, infrequent synchrony row or the intensive, static resource demand column.

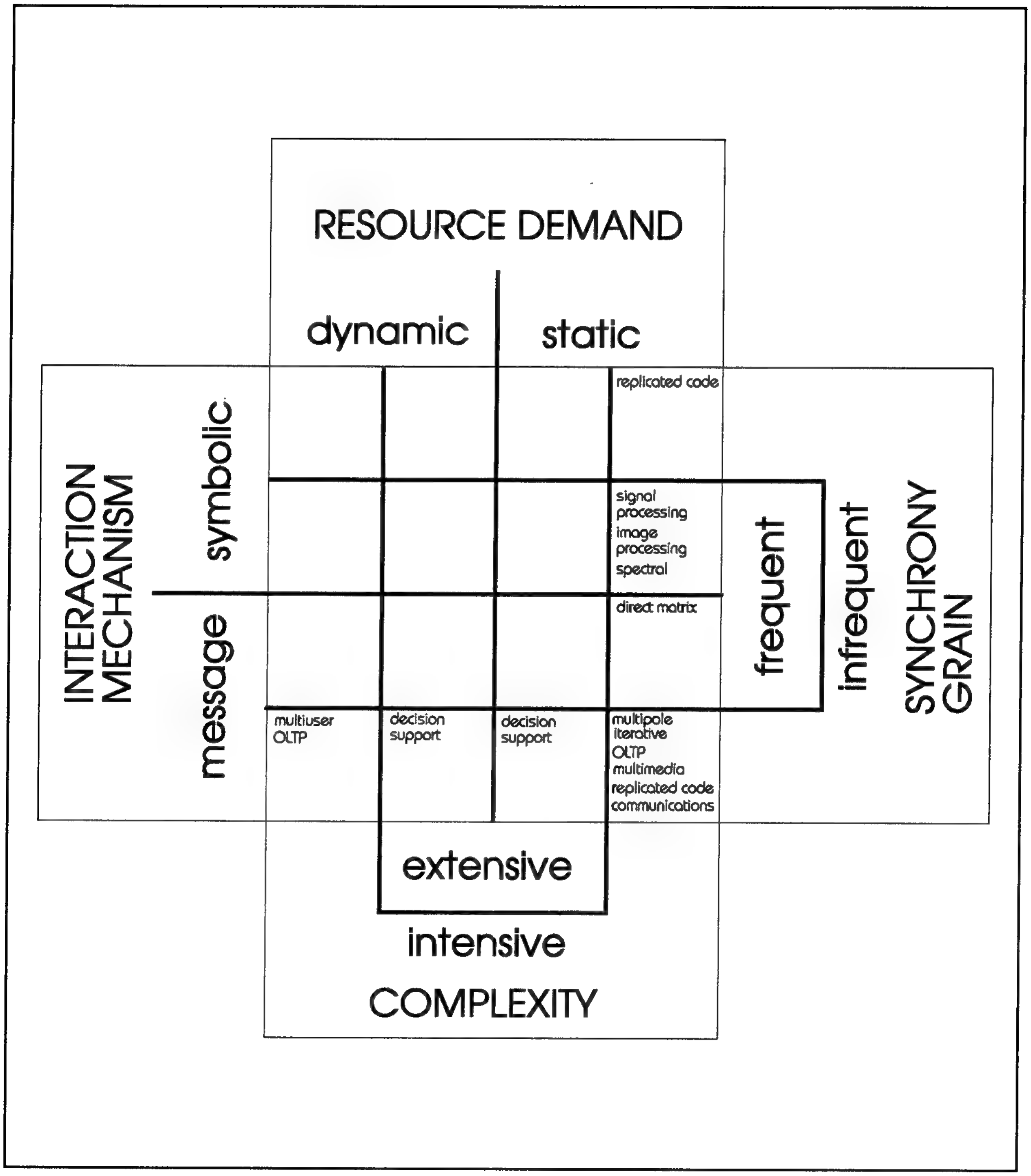


Figure 2 - 2. Character of Selected Applications

3 C^{3}I System Focus

3.1 The C^{3}I Requirement

Like many process control and alarm systems, the components of C^{3}I systems detect, track, identify targets, decide upon action, and take action. However, the C^{3}I system goes further. Commanders find that these systems give significant advantage when they distribute them across a wide or global area. It also includes components with forethought and planning processes. The system includes abstraction of enemy actions and responses. Its builders must include the following: Satellite sensors, flying radar platforms, radio signal senders and receivers, and a vast array of sensor platforms. These collect data and combine it into data bases to create a vast amount of information. Other parts keep a data base of conditions and state of readiness of weapons, sensors, personnel and supplies.

Projections into the probable future is a further feature. Commanders create battle plans from the C^{3}I system's intelligence data base. Those data bases reflect the present state of battle, projected supplies, personnel, and battle scenarios. Often the history of similar actions or battles is on line. Note that the time responses of the components vary significantly. Some require hard real time, where the system absolutely guarantees response time. Some require only very soft and casual timing. Often the extra iterations would improve plan quality by running more scenarios or iterations each day. These improvements will become common place when the capability is available.

Human interaction is common within the system, due to the complexity and uncertainty of decision making. Humans also make some decisions using only imprecise data and information. Errors due to unavailable, incomplete or imprecise information can lead to serious trouble. Errors often occur due to lack of information being brought to the human attention or to right location. As a result, humans make their decisions using information that is inaccurate or incomplete. This aspect of interaction with the system is one of the causes of complexity and dynamism of C^{3}I systems. Resolving this difficiency is is one of the opportunities afforded by general parallel systems that provide symbolic processing of the most difficult character.

Failures can make C^{3}I systems famous. Examples of this are: the destruction of an Iraq passenger airliner during the Gulf crisis or, the more recent example, when friendly helicopters were shot down when mistook for a Russian model. In both examples all the information was available somewhere in the system to make the correct no-fire decisions. However, the decision maker did not have it or could not use it. The individuals responsible typically defend their actions by citing their adherence to protocols, standards, procedures and training. All are parts of the "system" that software engineers must consider when they design and build.

Future C^{3}I systems are likely to add a significantly higher level of learning and adaptive methods to provide the information needed to avoid false detections. The delivery of information to the proper location or the removal of false information causes the parallel complexity challenge. The truthful and complete situation awareness capability will require parallel symbolic computational loads quite different from today's more structured numerical parallel applications. Thus, C^{3}I system builders face significant challenges for improving the capability of their systems. Parallel software engineering methods and improvement in the parallel industry infrastructure are needed to help in meeting this challenge.

3.2 C^{3}I System Overview

Therefore, a C^{3}I system is a large, dispersed, rapid response decision support system. The system includes global components that provide information for decision making by a hierarchy of commanders, leaders, and followers. Its information base changes frequently and contains incomplete and imprecise information. When active a C^{3}I system has a planning horizon is at most daily, and often hourly. It includes real time process control that recognizes, evaluates, notifies, and responds to external events. Once the system makes the decision to respond, the C^{3}I system can bring lethal force on any identified targets. As a result, its operating systems and application software must be extremely available, fault tolerant, reliable, and secure.

A C^{3}I system component works in a myriad of physical situations: in fighter aircraft, in large aircraft, in satellites, etc. Therefore, these components must provide their capabilities within rigid physical constraints of power consumed, heat produced, area and volume taken and electromagnetic emissions. Similarly environmental constraints have wide range of shock, temperature, humidity, and pressure. Since the C^{3}I system is subject to enemy attack, it must survive loss of parts with graceful loss of capability. Fault tolerance and rapid repair are necessary. The C^{3}I system is very complex, and is most likely built over many years. It is also likely to have an expected useful life measured in decades. It must be flexible to allow staff to modify it to meet new threats or unexpected applications. Developers also must consider that they might use the components in multiple higher level systems.

3.3 Parallel Computing and C^{3}I Systems

Due to the distributed and global nature of C^{3}I systems, designers place the capability for response into individual components or subsystems. Our focus is on how to carry out each component with parallel processors to reduce its computation time. The reduced time allows builders to significantly improve the C^{3}I system, by adding more decision making and information dispersion capability.

The budgets for C^{3}I systems require more commercial-off-the-shelf and high level coding content than used in the past. Faced with these burdens the builders of C^{3}I systems need both parallel computers and parallel software engineering processes and tools. The combination must give reasonable development and life cycle costs while delivering expected performance levels.

3.3.1 Effect of Technology on C^{3}I Systems

Ever increasing demands for higher performance is a common characteristic of large systems, both military and commercial. New missions and threats require continued performance increases and functional improvements within existing physical constraints. Excellent progress in workstation performance and human interfaces and standards (e.g., Common Operating System Environment or COSE) has led to higher expectations from the user. Now one can create complex requests and demand increased system capacity without realization of the demand's scope.

The potential leverage provided by parallel computing is significant for improving the performance of a C^{3}I system component. However, C^{3}I components often have complex and dynamic internal interactions. They may have the need to synchronize or communicate frequently and some have little recognizable structure. However, C^{3}I systems are also highly concurrent. The potential amount of parallel processing feasible within a given period is very high, in spite of its unpredictable nature. C^{3}I systems gain an advantage only if the parallel computers are capable and software is available for processing dynamic and complex parallelism. Note that numerical signal processing inputs are structured and regular. C^{3}I designers understand this type of intensive computation well. They have a less firm grasp of the symbolic parallel processing necessary to make complex and dynamic decisions.

3.4 C^{3}I System Limitations

The government makes significant investments toward the development of hardware systems, software applications and software tools for building and programming high performance parallel computers. So far these investments fail to address all the requirements of Air Force C^{3}I systems. Specifically they lack availability, real time response, security, fault recovery, reliable software, portability and reusability, etc. In addition, programming processes remain costly, difficult and non portable among different parallel computers. Users often find that high level programming only works for selected applications that consist of replicated copies and multiple independent tasks. They find reasonable performance efficiencies difficult to attain. System builders usually find a mismatch between the Air Force's dynamic and unstructured C^{3}I applications and the capabilities of today's high performance parallel systems. Commercial system builders find that their distributed servers are good matches only for multiuser "throughput" processing, where multiple transactions and multiple users run simultaneously. Their success has not yet been consistently replicated in complex business decision support systems with applications more closely attuned to C^{3}I systems.

Potentially, parallel processing offers significant operational speed and cost advantages for building C^{3}I systems. However, the present limitations of commercially available parallel computers restrict their use to static and well structured C^{3}I system components. These limitations also lead to expensive development costs and inefficient use of processing capability. In addition, the budgets for C^{3}I systems require more commercial-off-the-shelf and high level coding content than used in the past. Faced with these burdens the builders of C^{3}I systems need both more capable parallel computers and parallel software engineering processes and tools. The combination of parallel systems and software engineering processes and tools must yield reasonable development and life cycle costs and deliver expected performance levels.

3.4.1 C^{3}I System Profiles

The goals of C^{3}I systems are the following:

- ♦ Reduced Planning and Execution Cycle - Move Capability to Capacity - more iterations per day
- ♦ Decentralize Execution Control - Automatic and Human Local Situation Response
- ♦ Improve Joint Interoperablity/Joint/Common Evaluations - Global Views for Decision Making
- ♦ Distributed Command & Control - Communications of Plans, Situation Evaluations, and Actions

The computing technologies necessary for building Command and Control systems are:

- ♦ Decision support systems
- ♦ Systems engineering and integration
- ♦ Distributed computing environments
- ♦ Distributed data base management
- ♦ Human computer interaction

- Communications networks
- Intelligence exploitation
- Surveillance

The range of processing character in C^{3}I systems is large. An Air Campaign Plan guides the operations. The plan includes air battle planning, projected intelligence, battle plan execution, and enemy situation correlation. Those plans require simulations, historic data bases, and integration communications across the command structure. At each level, the users monitor, replan, direct and act upon the battle situation. Some plan elements change interactively, others hourly, others daily.

During execution, some components of the C^{3}I system operate automatically and respond to threats in real time. Others operate on the time frame of human interaction. They display information and interact with humans for decision processing. Surveillance systems collect and process data collected by many mechanisms across a wide spectrum. Multiple types of radar, radio interceptions, identification signals, optical detection, and human visual information are necessary. These signals are processed and integrated with intelligence information to provide a decision information base.

Some components communicate with other C^{3}I systems to obtain a global situation overview to guide subsequent battle decision making. Some actions follow battle plans made on a daily cycle. Often planners must adapt these plans and decision guidance on a short cycle. Ideally, a planner has multiple simulations of the proposed scenarios resulting from a plan. Data bases of assets and personnel are updated and used for planning. They are the basis for making resource requests and reports to higher level commands.

3.5 Application Character in C^{3}I Systems

3.5.1 Application Character for Components

Table 3 - 1 gives some processing types that match C^{3}I system computations. (Character criteria are defined in the Applications Section 2.) Figure 3 - 1 locates the character tendency of each application.

Table 3 - 1 C^{3}I Activity and Computational Components

C^{3}I Component	Mission Description	Application Characteristic (RTT = Response Time Type)
Signal Processing & Data collection - filtering and preparation	spectral, direct matrix, signal feature extraction, communication	RTT: Continuous Data Rate Concurrency High; Resource Stability Static; Synchrony Grain Frequent; Interaction Symbolic (Data Driven); Complexity Intensive

C^3I Component	Mission Description	Application Characteristic (RTT = Response Time Type)
Surveillance, target identification, friendly identification	symbolic feature extraction and interpretation, feature generation - hypothesis learning, feature extraction signal recognition, false signal removal, identification & false alarm detection	RTT: Response Time Goal Concurency Low; Resource Stability Dynamic; Synchrony Grain Frequent; Interaction Symbolic; Complexity Extensive
Air Battle Planing: Weapon resource allocation, air battle simulations, scenarios, projection of events, battle planning/history	planning and scenario evaluation, AI, data base, system decision making improvement (AI)	RTT: Capacity Constrained Concurency High; Resource Stability Dynamic; Synchrony Grain Frequent; Interaction Symbolic; Complexity Extensive;
Battle Execution: command response (rapidly varying demands on system), automatic response execution	real time, dynamic capability and fault tolerance system fault correction and recovery (system coordination with other components)	RTT - Hard Real Time Concurency High; Resource Stability Dynamic; Synchrony Grain Frequent; Interaction Symbolic; Complexity Extensive;
Human interaction	decision control, displays, alarm information presentation, human interaction (GUIs, transaction processing)	RTT Interactive Multiuser Concurency Dynamic; Resource Stability Dynamic; Synchrony Grain Infrequent; Interaction Message; Complexity Extensive;
Information Display	graphics, rendering virtual reality (scene projection & computation) sound and voice	RTT - Response Time Goal Concurency High; Resource Stability Dynamic; Synchrony Grain Infrequent; Interaction Message; Complexity Intensive;
Battle actions	potential event recognition (multidimensional correlation, blackboard), event determination (numerical, multipole, symbolic), decision surface computation (neural networks, AI), multiple event association and data fusion (fuzzy logic)	RTT: Response Time Goal Concurency High; Resource Stability Dynamic; Synchrony Grain Frequent; Interaction Symbolic; Complexity Extensive;
Battle Commands	weapon system command, human overview, command overview	RTT: Interactive Single Job Concurency Low; Resource Stability Dynamic; Synchrony Grain Infrequent; Interaction Message; Complexity Extensive;
Battle replanning: battle resource limits, constraints and projections	linear algebra, resource modeling, data base access	RTT: Capacity Constrained Concurrency Low: Resource Stability Static; Synchrony Grain Infrequent; Interaction Symbolic; Complexity Extensive

C^3I Component	Mission Description	Application Characteristic (RTT = Response Time Type)
Situation Evaluation, position and geometry correction	filters, transforms, spectral, real time decision processing, data base	RTT: Interactive Single Job Concurrency High: Resource Stability Dynamic; Synchrony Grain Infrequent; Interaction Symbolic; Complexity Extensive
Communications, external information exchange, joint operations and multiple users	communication switches, decision making, transaction processing, data base, external oversight, signals via IFFN system	RTT: Response Time Concurrency High: Resource Stability Dynamic; Synchrony Grain Infrequent; Interaction Message; Complexity Extensive

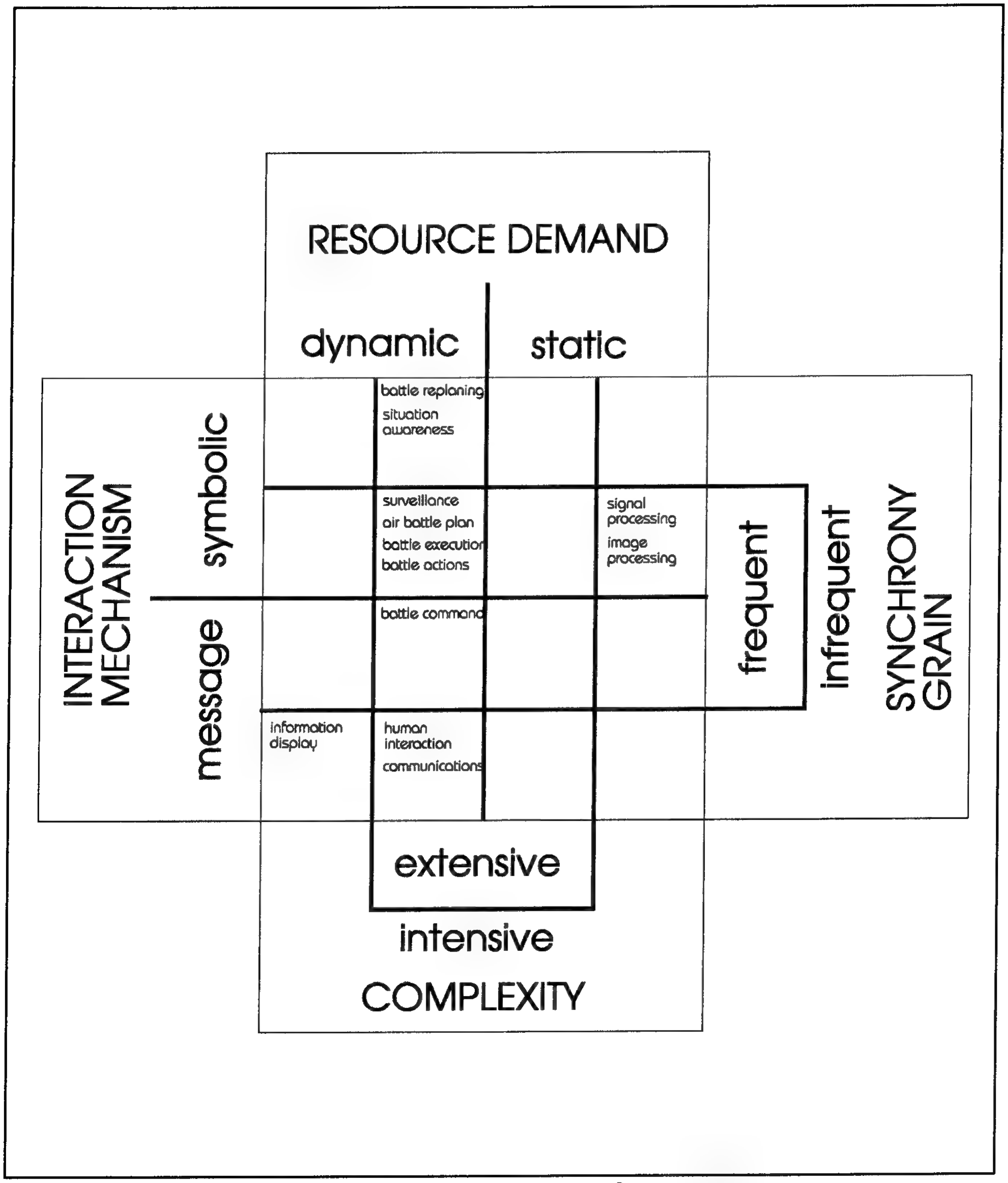


Figure 3 - 1. Character of Various C^3I Applications

4 Parallel Architecture Assessment

4.1 Parallel System Types

C^3I systems include application components of various time response types and physical constraints. The optimum system design might include different parallel architectures of many organizations and types to best fulfill mission needs. The following parallel system types may be necessary in C^3I systems:

- Capacity-scalable Multiprocessors (constant bandwidth capacity between any two processors for any processor count)
- Cost-scalable Multicomputers (constant bandwidth capacity added per node)
- Symmetric multiprocessors (bus limited with shared memory)
- Networks of computers (clusters)
- Multiserver systems
- Heterogeneous computing

Appendix A. gives the definitions of these systems as they relate to C^3I systems.

4.2 General Assessment of Parallel Computing

4.2.1 General High Performance Architectural Trends

Shared memory vector supercomputers still set the standard for production supercomputing. They provide extremely high individual processor performance and a processor count of sixteen. Most often, they deliver a higher fraction of their maximum performance than other parallel architectures. The large-parallel vendors are attempting to move toward production systems by lowing their maximum processor count and paying more attention to provisions for input/output and multiuser access. As a result, they now use the label "scalable-parallel "instead of the "massively-parallel" label. Vendors of scalable parallel system now place greater importance on system robustness to make their move to production computing. Shared memory symmetric multiprocessors with low processor counts are very successful commercially as throughput servers. Clusters are now available for replicated and multiuser throughput applications.

4.2.2 Microprocessor Trend

Vendors of most parallel machines most often use the same microprocessors as found in workstations. Using those microprocessors as the processor in parallel systems is believed to keep a vendor on the latest technology curve at a low cost. Those who wish to cover large-scale multiprocessor, cluster, and symmetric multiprocessor markets gain a standardization advantage. Vendors also may not have to develop all of the individual node operational software. Application vendors are also more attracted a familiar processor when porting. In addition, microprocessor vendors are taking a more active role in symmetric multiprocessor designs and clusters. Table 4 - 1 gives microprocessors that are used in selected parallel systems. The trend is not universal. nCUBE maintains that a special purpose processor with integrated interconnect paths is a better path to high availability and robustness.

Table 4 - 1 Microprocessor Application to Parallel Computers

Micro-processor	Parallel Computer	Type	Interconnect Structure	Comment
MIPS	SGI PowerChallenge	Multiprocessor	Bus based Symmetric shared memory	(18 most powerful nodes or 36 with less powerful)
PowerPC	IBM SP-2	Multiprocessor	Cross Bar	Large processor counts
Alpha	Cray Research T3D	Multicomputer	(3-D mesh)	Tightly Coupled to Vector Super Computer
Alpha	Digital Equipment Corp.	Multiprocessor	Bus based Symmetric shared memory	
Precision	Convex Computer (Exemplar)	Multiprocessor	Both Bus Based Symmetric and SCI directory based shared memory	
SPARC	Sun Microsystems and Cray Research	Multiprocessor	Bus and Multi-Bus Based Symmetric shared memory	(Sun 20 and Cray 64 maximum nodes)
SPARC	Thinking Machines	Multiprocessor	(a limited fat tree)	SPMD and SIMD operation
Pentium	AT&T (NCR)	Multiprocessors (2 types)	Symmetric shared memory to 32 nodes, Crossbar based above	Software tools across both architecture types (OLTP and DBMS)
i860	Intel	Multicomputer	2-D mesh	Large maximum count (not a workstation microprocessor)
Pentium	Compaq, Encore, Gateway, Dell, IBM, AST Research, Hewlett-Packard, etc.	Multiprocessor	Small Processor Count Symmetric shared memory for Network Servers	UNIX and Windows NT Bus Chip sets: Corollary, LSI Logic, Pequr and Wyse
in-house	nCUBE	Multicomputer	(hypercube)	Large maximum processor counts

4.2.3 Processor Count Trend

4.2.3.1 Large System Maximum

Scaling of an architecture to Teraops (10^{12} operations per second) peak performance is a feature necessary for only a few systems in very few locations. Focusing the design of cost-scalable multicomputers to meet those needs often compromised performance at a lower processor count. This sacrifices volume sales in a larger potential market. Application scalability over a range of ten to a few hundred processors is more important than providing thousands of processors. Without an effective and economic low processor count capability, market volume lacks the gross margin necessary for building operating system and other system software. Sales volumes necessary to achieve critical economic mass, require volume at lower processor counts and demonstration of high processor count capability. The latest large-scale architectures now trend toward better performance at reduced maximum processor counts - tens to hundreds not thousands.

4.2.3.2 Symmetric Multiprocessor Processor Trend
A second trend that has significantly more economic power behind it is following this sequence:

- Start at a very low count, as low as two to four
- Convince users of its value
- Incremental build software capability
- Increase processor count as hardware interconnect technology allows

The vendors of workstations and personal computers have taken that approach. Competitive designs are coming from symmetric multiprocessor vendors using high performance bus systems. Once limited to four to eight processors, symmetric multiprocessors now go to sixty-four processors (Cray Research SuperServer) using four complete cache coherence busses. AT&T Global Information Systems sells server systems with data base and OLTP applications. Their models scale across symmetric and tightly coupled distributed multiprocessor architectures over the range of two to 512 processors.

4.2.4 Interconnect Latency and Bandwidth Trend
The cost-scalable multicomputer vendors trend to the optimum point on the connectivity (the number of network routing paths from a given point) and bandwidth curve. Now under the name of "scalable" parallel processing, this trend is changing. A generation back, researchers favored the hypercube and other high connectivity networks. Then designers began to reduce the connectivity and increase the bandwidth of individual links. Their reason was that the communication mechanism - message passing using large messages - fits well with low connectivity interconnects. Delays in the protocols for creating and transferring messages dominated the latency. Engineers are reversing the trend by reducing protocol delays. Examples of the trend reversal are IBM with eight way cross bars, and Cray Research with a 3-D mesh. These changes stress low latency for small packet transfers. nCUBE's use of an eighteen-way path is useful in the multimedia server application, another indication of a reversal in this trend.

4.2.5 Network Bandwidth Trend
The addition of fiber channel standards for local area networks and high performance National Infrastructure optical fiber networks will introduce a new factor into the trends for multicomputers. The expected result is that there will be a well defined split in the systems used to support applications of different character. This trend will also change the designs of C^3I systems. Designers will likely identify components differently if this bandwidth capability is available. In addition, the higher network bandwidth capability creates a problem for workstation that must match the network's capability to deliver data.

Such high bandwidth capability provides the capability necessary for intensive, large-message interacting, infrequent synchrony, and static resource applications. Many applications with that character may transition directly from multicomputers into very high performance clusters. These applications internally use large messages to cover protocol latency. Many will advocate that this changes the balance toward use of multiple workstation clusters and away from multicomputers for capacity constrained and multiuser interaction applications. The present successful application character types on multicomputers will turn to the cluster as a more cost-effective means of computation.

However, the impact on more dependable time-response-goal applications will be small. There will be little impact on applications with dynamic resource demand, frequent synchrony, symbolic interaction, and extensive complexity character. These require a close coupling and latency hiding not feasible on the workstation cluster or multicomputer.

4.2.6 Capability Trend

Vendors have made significant increases in input/output capability and disk support to make their machines more practical for continuous production operation. A glowing success of parallel systems is the capability to provide high throughput for transaction processing systems. Pyramid, Tandem, AT&T, Sequent, and Encore are companies that do business in this important application area. However, their systems face a barrier to performance on complex query solutions because they have limited capability for path changes during execution. (Sequential based tools now statically aid complex queries.) A parallel intelligent decision system approach, one that dynamically issues new query paths, requires an advancement of machine capability beyond today's throughput processing.

4.2.7 Cost Trend

Cost per peak speed trends are up for multiprocessors. However, performance delivery is improving in some models and cost per delivered performance may be down. To meet quality necessary for production environments, there is a necessity to increase interconnect performance, provide fault tolerance, develop robust operating systems, and provide the input/output balance. Paying for this requires added costs. However, good designs should deliver more effective performance with the added capability.

The foundry cost differential between using a specialized processor instead of a standard workstation microprocessor is not the significant cost benefit factor once believed. Interconnect costs are necessarily high when vendors serve a broad application spectrum. System software costs have been significant. Reliability and instability problems have plagued some scalable architectures. The conclusion is that good computers are expensive to build. Vendors still target designs of today's machines to particular markets. This lack of support for a broad application set has several disadvantages. It leads to a limited market and does not allow for amortization of either system or application development costs across several machines.

4.3 Related C3I Factor Trends

4.3.1 Programming Model Support Trend

Features have been added to the interconnect/backplane to support either shared memory or scalable parallel message passing communications between processors. The shared memory programming models have cache coherence components that allow cache updates when any processor writes a location. Dedicated hardware that transfers buffers between processor memory (e.g., IBM SP2) also directly supports message passing. Meiko supports the Computing Surface model in a VLSI interface circuit at each processor interface to the interconnect. AT&T provides its commercial software tools for decision support DBMS access on small symmetric multiprocessors and on its large-scale parallel machines. The largest of these machines emulate shared memory to ease programming effort.

Meiko, IBM, Cray and Convex have recently introduced machines with global virtual memory, more robust and reliable interconnect designs and more balanced input/output capability. IBM, Cray, and Convex have made heavy investments in complete solutions, including commercial business and production technical design and engineering applications from independent software vendors. The Scalable Coherent Interface (SCI) bus used by Convex also has a large processor-count and a directory-based-cache-coherence mechanism. They, by necessity, have concentrated on higher reliability and programmability than the early research machines. If they succeed in meeting commercial availability and programmability standards, they have the potential for widening the acceptance of parallel computers. This could convince large numbers of independent software vendors to begin to invent and provide new parallel applications.

If this trend continues, the small message latency in multiprocessor machines of equal size will reach a few hundred ordinary instructions. This will make the distinction between small message passing and shared memory operations less visible. Both require a formal protocol for sharing objects. Designers can improve both by increasing local store for application context and by providing fast hardware multithreading between accesses and memory. Features from research computers such as the MIT and Motorola *T and the Horizon Tera will be added to make a more suitable general purpose multiprocessor.

4.3.2 Obsolescence Trend

There is a flux in performance and architectural designs on the market. Because of vendors' concentration on research markets, yesterday's machines emphasized peak performance instead of reliability and programmability that is necessary in the commercial marketplace. As a result, significant government direct and indirect funding is necessary for their existence. The risk to the government is that many programs committed to those machines cannot take advantage of software from a broad base of suppliers. This cancels the anticipated COTS advantage. As a result, C^3I systems builders cannot take advantage of a broadly accepted programming model and tool sets that meet all their need.

4.3.3 Availability Trend

The latest trend is to add hardware and operating system features for increased availability. AT&T 3600, with folded Banyan interconnect, has built in redundancy control for fault tolerance. The folding provides lower latency when fully functional. One failing of research oriented parallel computers is that both hardware and operating system reliability have fallen short of that required in both engineering and commercial processes. Since the most common use is outside a mission critical application this deficiency survives. Industry's trend is to correct this shortfall. Adaptation outside the research computing environment requires high availability.

4.3.4 Real Time, Security and Physical Constraint Trends

The capability to execute acceptably within hard real time systems is not one that scalable parallel computers have reached. This area remains a specialized parallel processor field with corresponding high life cycle costs and long term support issues. Some developers are building hardened and repackaged versions of parallel research multicomputers. These developers are now discovering that they require a special operating system kernel to meet real time goals. A guarantee of time response requires capabilities far from those in a computer designed solely for capability-constrained and multiuser-interactive applications. The real time result has little resemblance to any software from the scientific research community. Therefore, there is too little technology spin-in from the commercial environment for the government's benefit. Turning a scientific research machine into a real time one may be more difficult than building a real time capable one from first principles.

4.4 Architectural Trend Conclusion

The shift toward production from research is leading to the following technical trends in parallel computers:

- Use of workstation-compatible microprocessors
- Convergence of processor count
- Increased interprocessor bandwidth/ decreased latency
- Programmability enhancing mechanisms
- Better balanced input/output, disk, and memory subsystems
- Improved fault tolerance, reliability, and decreased repair effort and time

The latest market introductions have punctured the idea that cost-scalable computing is generally useful. High performance clusters are threatening those architectures for large-message passing applications. Capability-scalable machines deliver a better cost- to-effective-performance ratio than cost-scalable ones. The extra investment in interconnect structure offsets lower software costs and wider applications set and market. The extra hardware and system software for fault tolerance are necessary for the commercial market. Vendors should find that satisfied clients return these costs by purchasing more goods.

Machines are now more capable but with fewer processors and lower peak speeds. Higher and more stable communication bandwidth capacity between processors is the result. Vendors are putting hardware support parallel commands and programming models into practice. Availability increases because fault tolerance features must meet commercial user needs. Some data base and transaction type applications have been ported across a range of symmetric multiprocessors and distributed multiprocessors. The mix of architectures is still too diverse. However, the trend to increase interconnect capability reduces the wide differences between architectures. These differences become less a disadvantage to software engineering processes for parallel systems. This is because adequate capability gives faster parallel instructions over a wider range of applications. Thus, parallel interaction timings are ratings of the suitability of a machine for an application. Then, the architecture's structure can be hidden and a quantitative evaluation of cost-to-performance benefit comes from the parallelism content of the application.

Providing compatible software tools over a wide range of processor count and architecture pays off, as shown by the sales success of AT&T for its OLTP and DBMS machines. As capability-scalable machines grow in the commercial market the benefits of large scale markets will provide better tools and broader understanding. This doesn't mean that these trends will solve all the problems of parallel programming because today's parallel computers remain overspecialized. It only means that it now may be feasible to accomplish some ideas and tools necessary for portability and low-risk design.

5 Market Forces and Issues

5.1 Market Overview

5.1.1 Active Market Components

Parallel computing has achieved several high returns on investment successes. Business firms have reported excellent results with parallel on-line-transaction-processing (OLTP) and decision support system (DSS) applications. (See the CAPPS '94 Conference, October 17-19, 1994, Austin, Texas.) They have reported high return-on-investment for these data-base-management-system (DBMS) applications. Commercial information system builders like the "no brick wall" effect provided by modular growth and large processor counts given by parallel computers. Others have found improved quality and engineering productivity by spreading parameter sets among many program copies. Industry's use of "replicated" parallel computing allows evaluation of design choices under simulated environments. The government's Grand Challenges have shown that scientists can expand their problems and speed up their solutions if they find suitable algorithms. Parallel computers work for research scientests. Some scientific applications, once thought to be unsolvable, are now within reach of researchers.

5.1.2 Industry State

However, these successes have not yet engendered a robust industry. Today's parallel systems fail to meet many needs. The parallel successes are only for data intensive, structured, statically allocated applications. These include the following: OLTP, a segment of DSS, replicated engineering, and Grand Challenges. They fail to meet the complete needs of command and control systems, real time control systems, symbolic decision making, and other time response critical applications. In addition, the limited suitability of large versions of these machines makes system building and life cycle support expensive and difficult. There are few systems or software engineering mechanisms that let designers design-to-deadline, predict reliable project budgets, schedule and estimate performance.

5.1.3 Cost Model

One reason for this is that the industry has fallen into an oversimplified cost model. Too often, academic and government researchers base their purchase on a cost evaluation, which declares that integration, software and support are essentially free. This leads vendors into a peak-performance-at-lowest-cost strategy. Too often, other users follow this lead and evaluate hardware in terms of peak performance per unit cost. Some business purchasers have a better model. They buy a specialized machine based on transaction or database access rates in an established benchmark. However, both market groups frequently fail to consider programming, integration and life-cycle advantages. If considered a more general purpose parallel machine would rate higher. They chose, instead, one that can execute a very limited function. This requires integrators to build complex structures around the computer whenever it is ineffective on an operational requirement. Therefore, they expect vendors to provide rock-bottom hardware prices and accordingly reduce interconnect and executive software systems to their minimum. Then they pay the price in integration and services to implement a reasonably effective solution to their application.
These peak-per unit-cost performance numbers drive the competition. Therefore, evaluators fail to make the costs of applications programming, integration within a business or supercomputer facility, and life cycle

maintenance of applications a significant factor. Purchasers have given little incentive to vendors to create an easily programmable computer if that computer has added hardware costs. There has been an infrastructure development failure due to incorrect economic models of parallel computers. The lack of knowledge about the best way to evaluate the economics of parallel computers comes from poor leadership in high performance computing circles. A commercial off the shelf strategy for parallel computers cannot succeed until this problem is corrected.

5.2 Market Driven Technology Forces

A successful strategy for C^3I system and software engineering must be consistent with, and take advantage of, the market power of commercial information and technical engineering systems. The strategy must consider dynamic changes in computing paradigms. In this time of rapid change C^3I system builders need to find a strategy to gain advances in parallel computing through market shifts. Real time, fault tolerance, security, and large scale symbolic decision applications are also important to business and technical computing. If C^3I system builders can show the way, parallel computers may provide significant capability advantage and avoid today's high development and life cycle costs.

5.2.1 Opportunity and Challenge

The challenge to the C^3I system builder is to encourage a market for parallel COTS products. To attain a market size suitable for attracting the independent software vendors the hardware products must be adequate for a wide range of applications. The software must be portable and effective across a wide range of architectures and machine sizes. This may result in increased hardware costs because the vendor must beef up the interconnect and improve fault tolerance and recovery capability. Robust interconnect performance and machine availability is necessary.

Today's research parallel systems do not meet all the requirements of the production center for technical or commercial computing. The available programming models yield neither portable nor scalable applications.[5-1] Aggressive effort for commercial decision support systems often requires development teams of tens to hundreds of people. (In spite of these high costs, advocates still report a high return-on-investment.) A lack of software engineering processes to deal with system and parallel computers often leads to inefficient and ineffective projects. Much of the present parallel computer research community considers cost-scalability of hardware above the need for application scalable software.

5.3 Dynamic Market Conditions

5.3.1 Expectation Expansion and Advances

The human interface is changing rapidly. Object oriented screen building has reduced the cost of conventional interface construction and human interaction. Voice commands, pen control, written text to ASCII character transformation, and wireless and remote, self-contained information collection capabilities are now available. Network control and communication's software for building homogeneous clusters of computers is now feasible for Capability and Capacity Constrained time-response-goal applications. Technology advances in

[5-1] Portability in parallel computing requires both effective execution and consistent operation.

field programmable devices, magnetic storage media, and virtual reality add to the opportunity for computer technology's capability to contribute to C^3I system capabilities.

5.3.2 Demand for Commercial Parallel Systems

The 1993 New York Times best seller, "Reengineering the Corporation," by Hammer and Champy, describes a revolution occurring in corporate organization. Corporations are rethinking fundamentals and making radical changes to achieve dramatic results. The key to that revolution is the identification of "value added processes." Hammer and Champy emphatically make the case that all large United States corporations must undergo "reengineering" if they are to remain in the worldwide, competitive marketplace. Many major US corporations are now committed to reengineering. The key to success is to use information, simulation, communication, computer information technology to build the new process capability. The level and direction of the "Reengineering" depend upon available and projected information technology. A significant market is emerging for parallel computers to support the needs of corporate reengineering.

Workstation vendors play a major role in the market for parallel servers. This commercial trend reveals that hardware vendors are expecting rapid expansion in the use of symmetric multiprocessors. These computers operate well in a throughput mode where several users share a processor. Symmetric multiprocessors are not subject to message-passing mapping effort. However, they require threads programming, using locks and semaphores. A suitable technique in small projects. However, in large projects this technique requires formal protocols and very careful programming to avoid concurrency conflicts. Independent software vendors are not enthusiastic over use of proprietary threads. They question the portability with effective performance promise of POSIX standards since vendors still tout their proprietary organization of threads, yet claim POSIX compatibility.

5.3.3 Large Scale Commercial Parallel Systems

These dynamic and fast-moving trends should create a market for new multiprocessor software products. These new products create the conditions for additional hardware and software products. Synergism like this creates a paradigm shift. If this paradigm shift is to include the larger capability-scalable multiprocessor systems, they require a programming model that allows them to effectively execute the same source-level code. The steps for consistent operation and effective operation on a different problem size and different machine should be similar to workstation technology portability. Consistent and effective execution of scaled-up symmetric multiprocessor applications on a capability-scalable multiprocessor server is the desired capability. This capability allows development on a small scale and production on a large scale.

5.3.4 Limited Commercial Applications Successes

The reason that multicomputers are not ready for large scale decision systems in commerce is that they are too limited in application type. [Richmond: July 1993] The introduction of large processor count multicomputers and multiprocessors into commercial uses is inhibited by the following reasons.

- performance is uncertain
- difficulty of carrying out the project
- interaction with heritage systems and interface is difficult
- software tools are not in place

- obsolescence leads to tactical applications instead of strategic
- high implementation complexity leads to high life cycle costs
- cost of project raises visibility to high-level management
- economics allows only incremental changes, not radical ones
- innovation and dramatic gains are too difficult to create

Uses of today's OLTP and DSS computers are limited to applications that are not of the most demanding character (extensive, dynamic resource demand, frequent synchrony and symbolic interaction). Humans must intervene and guide the interactive cycle of queries and decision interaction now demanded by large decision support applications. Adaptive intelligence methods could do the intervening human interaction if the computers could effectively serve these difficult application types. System builders are expecting software to break this barrier. However, without the underlying machine and operating system capability for the most demanding applications, we cannot expect significant market impacting progress.

5.3.5 Technical Computing Marketplace

Computer science research has failed to develop in a direction that fits it into the production engineering process. A concentration of grand challenges has lead the field away from the industrial processes and operations necessary for market acceptance. Typical engineering projects are performed using workstations and low processor count servers reached via a network in a client-server mode. Their present application acceleration direction will be to add small symmetric multiprocessors into their network. Instead of concentrating on creating an integrated industry, parallel programming research has abandoned many practical engineering process applications and concentrated on those requiring Teraops level processing speeds. The result is the absence of an infrastructure that understands the transition of parallel applications into the engineering process.

5.4 Commercial-off-the-shelf (COTS) Parallel Computers

5.4.1 Strategy for Future Cost Effectiveness

Commercial-off-the-shelf (COTS) components are important and provide an available technology for building high performance C^3I systems. The economy of scale of having commercially available hardware and software leads to widespread use of COTS workstations and associated technology. Workstation operating systems, data base systems and high level languages are readily available for scientific and commercial uses of workstations and personal computers. The success of workstations and their ever increasing performance and programmability leads to the same high expectations for parallel systems. These strategies are valid for workstations and personal computers. However, the COTS strategy has not worked for parallel computers.

Before a COTS strategy, C^3I system designers often developed special purpose parallel computers to meet specific mission needs. Each had its applications carefully mapped to achieve high performance. Designers isolated these specific capabilities. They placed them only where needed and where structure matched needs. These special-purpose, parallel computers had specific, well-engineered capabilities. The development and life cycle costs were high but they usually met stated performance and physical constraint goals. A successful COTS strategy would significantly reduce these development and life-cycle costs. However, designers find that vendors have tuned today's machines for specialized operations that do not match all C^3I system needs.

More important, because there are no general parallel machines, designers have not created a mission vision that significantly advances C^3I systems capability as far as they could have. What one does not attempt is the greatest loss. There are important parallel C^3I system applications that designers have not attempted because their feasibility is uncertain on today's parallel architectures. The same is true of business and engineering processes. Computers that allow only well structured applications severely limit the vision of computing. They reduce innovation. The parallel computer industry needs to eliminate the burden of feasibility uncertainty for applications that do not match the specially tuned product.

5.4.2 Generality Failure Impact

A successful parallel COTS approach offers the advantages of common and widely used tools, software, computers, and peripherals. However, attempts at a common programming model have been limited to well-structured applications, not general ones. One reason the strategy has not worked is that advocates must treat today's scientific-research parallel systems as specialized processors and not as general computing products. Their applications are typically carefully selected to match the limitations of the parallel computer. As a result, research departments use them, not the engineering production department. This limits sales volume and product acceptance. A market that is limited to a few hundreds of units does not engender a good strategy for attracting and building a COTS industry.

The failure at generality forces mapping and specific interaction between hardware and software. That leads to non portable codes. A key feature of the success of workstation software is that applications could be easily ported between vendors without detailed regard for specific hardware machine details. Hardware vendor engineers do not need knowledge of the software vendor's applications in any specific way and vice versa. Each goes on with assurance that they can attain a reasonable performance level without tying themselves to a specific matching technology. Without independence of hardware and software developer there cannot be a successful COTS strategy for parallel computing.

Since there is no general programming model, each parallel industry participant faces an added development and economic burden.[5-2] Independent software vendors find less than a critical mass. They cannot amortize software development costs over thousands of sales. Hardware vendors fail to achieve their promises of hardware cost saving. Hardware is often obsolete when operating systems mature. Vendors and their clients have trouble in attaining desired performance and functionality. Clients must risk high development and integration costs and long term development. Finally, the products do not meet the demanding application's mix and integrity expected by consumers.

5.5 Strategy for Embracing the Commercial Successes

The opportunity for the Air Force is to recognize the shortfall in proprietary threads and large-message passing models. Then define a standard programming model that will work on both symmetric multiprocessors and low latency capacity-scalable architectures. Capacity-scalable parallel processing

[5-2]This statement implies that PVM and the MPI are not general programming models. They are message passing models which gain effective portability only for selected, mappable applications . Since their programming model cannot effectively model applications with symbolic interaction, extensive complexity, and dynamic resource demand character, it does not meet general application needs.

applications and symmetric multiprocessing should be in the same software applications and tools market. Independent software vendors would then find the opportunity for profit from parallel computing. This produces a larger base for the creation of software engineering tools and processes. The Air Force goal should be to create a compatible tool and model standard that applies to the large capacity-scalable multiprocessors required for dynamic, complex C^3I systems. The result would be programmable scalable parallel software applications that span architecture types and sizes.

6.1 Parallel Software Engineering Goals

The goals of parallel software engineering are the following:

- Reduction in development risks and costs
- Reduction in life cycle costs
- Separation of hardware and software development
- Portability combined with effective performance
- Elimination of application suitability uncertainty
- Predictability of design and performance
- Robust interaction mechanisms with heritage systems
- Increased capability to make innovation and dramatic gain
- Mechanisms that allow capability for real-time, secure, fault tolerance, reliability, etc.

Parallel software engineers cannot meet these goals without a suitable machine foundation with effective execution capability for all application character and time-response-goal types.

6.1.1 Need for C^3I System Software Engineering

The parallel industry lacks a foundation for engineering C^3I components. That deficiency results in less innovation and aggressive design, leading to weaker C^3I systems.

Software engineering deals with large, complex, dynamic, mixed hardware systems that have a long useful life. Parallel research technology now exacerbates the software engineering process. Performance and programmability on large-scale, complex, concurrently operating components are issues important to C^3I systems. Developers of such systems require multiple organizations, large programming teams and a hierarchy of skills and knowledge. C^3I systems operate and change over a long period -- the life cycle. The systems are often multiserver and contain a mix of computer types ranging between special developed processors and commonly available commercial workstations. The systems often operate over a wide geographic range within widely varying environments and physical constraints. Their components may be space, airborne or at fixed locations spread across a wide area. Annex A - Appendix B looks at Software.

Designers select the capability and a time-response-goal for each component in the C^3I design. They must design to guarantee that some of these components complete within each execution frame. The components also vary in character. Components may be the least difficult: intensive complexity, infrequent synchrony, large message passing interaction and static resource demand. Or they might include the most difficult: extensive complexity, frequent synchrony, symbolic interaction, and dynamic resource demands. Designers also assign each component its mission related C^3I factors. These include availability, fault tolerance, security, safety, physical constraints, etc. Software engineering processes provide a rational means of dealing with these difficult systems. For parallel computer components, there is a lack of a robust parallel foundation and supporting engineering processes and tools. This deficiency intensifies these conditions to the point that C^3I designs are less aggressive and inventive. C^3I engineers may only attempt applications with the least difficult parallel application characters.

6.1.2 Rational Engineering Process

The tools and capability for a rational engineering process for parallel software engineering are not available. The result is higher development risks, costs and time.

A rational engineering process requires quantitative prediction about both product quality and process. A product quality measure matches functionality, integrity and performance to requirements. The process measures are effort, skills, cost and schedule. However, system designers often state requirements and environmental factors uncertainly. In addition, requirements can change after delivery because buyers recognize new mission opportunities. In addition, unless incremental progress is demonstrated early in the program, management support and enthusiasm can change. Therefore, developers need a fast development cycle. In a rational engineering process, designers use a progression of iterating stages within incremental builds to get a better product. Each stage and build can be more accurately forecast and engineered because smaller steps reduce risks. Separation of process engineering from technology adoption and change is necessary for rational engineering [Murphy 94].

6.1.3 Demand for Quick Results from Complex Applications

Higher expectations due to acceptance of interactive human interfaces are leading to higher performance expectations. Parallel systems are a potential solution to this demand.

Ever increasing demands for higher performance is a common characteristic of large military and commercial systems. New missions and threats often require continued performance expansion. Recent progress in workstation performance and human interfaces have lead to higher expectations from the user. Now she can easily create complex requests and expect the same rapid response as prior simple screen constrained requests. This higher expectation leads to demands for higher performance and new operational actions and responses. The 35% growth in processor speed each year of workstations and microprocessors provides one performance deficit solution. However, many systems now have performance gain assumptions already built into the system design. Parallel computers are one solution to these performance demands.

6.2 Parallel Process and Design Metrics

One major weakness in parallel software engineering is the lack of a staged design process for assuring performance expectations of parallel computer-based components.

Ideally, the tools to support performance delivery and program goals would be based solely on a set of application characters, time-response-goals, and C^3I mission factors. Engineers should have tools that, based on the application character and time-response-goal, can provide performance simulation and evaluation.[6-1] Once the designer gets performance risk estimates, there should be tools and metrics to estimate the required schedule, skills, cost and labor. There are now no widely accepted process or metrics that provide quantitative performance and program assurance of this form.

[6-1] Rome Laboratory's PAWS is an example of an application character to architecture tool. However, it presently is little known outside a narrow group of Syracuse University (the developer) and Rome Laboratory user.

6.2.1 Quantitative Performance Assurance

The industry lacks a method of staged performance evaluation. Performance on a C^3I mission means effective delivery of character functions, time-response, integrity, and physical constraints.

Designers measure performance against the functional, performance, integrity, and physical constraint requirements of the C^3I mission. System designers include factors such as the following: availability, physical constraints, real time response types as parameters in performance requirements. Ideally, software engineers can define the characteristics of an application based on the observed characteristics of similar systems. After defining those characteristics it should be possible to use a set of standard parallel instructions to predict the performance of the application on different parallel architectures. A software engineer needs a measurement scheme for testing and responses of how well architectures deliver operational speed and matches the C^3I mission factors according to prediction.

6.2.2 Quantitative Program Assurance

Designing the development process before starting is as important as designing the product before building it. The industry lacks the rudimentary information necessary to rapidly develop parallel systems on fixed resources, physical constraints, skill, cost and time budgets.

Development program goals are measurements of skills, schedule and cost budgets. A tool should give developers a metric for difficulty of development based on the application and on a selected machine. This tool would provide the ability to make choices about the allocation of costs to hardware or software development. An output should be a cost, labor and schedule estimated based on skill levels of the development team and Software Engineering Institute rating of the project's management process.

6.2.3 Development and Life Cycle Metrics

Cost per peak performance models are inaccurate because they fail to include application software development and life cycle costs.

An ideal measurement of a parallel computer is to evaluate its impact on the system's development and life cycle costs. Parallel computers potentially eliminate many other computers in the system. Their high execution rates can make a system feasible and cost effective. However, if the cost of software development increases to eliminate hardware cost savings, then the parallel computer loses its cost effectiveness. Comparisons of parallel computers by evaluation of mission life cycle costs could be telling factors in an evaluation. In general, those systems that are the most quantitatively predictable and programmable have the lowest support impacts from assumed mission changes. Evaluators set criteria from mission need statements. They need a process for relating mission need and life cycle costs for alternative parallel solutions.

6.3 C^3I Factor Evaluation Metrics

The industry lacks a set of parallel rating criteria with respect to the following: capability for supporting a range of time-response-goals, effective performance for mixed application characters, and C^3I integrity and constraint factors.

Engineers use more than speed as performance criteria to judge a system. Consequently benchmarks include measures other than speed measures. The suitability of a parallel computer for a particular system depends upon many criteria. Commercial organizations insist upon proven availability and portability before even considering speed performance factors. The following are evaluation characteristics for parallel computers:

- Availability includes stability, reliability and fault tolerance of both the hardware and operating system
- Programmability includes the following: portability, reusability, relative development ease, consistency with normal programming methods, and performance prediction based on software measurements
- Physical Constraints includes power, weight, volume, sizes, electromagnetic noise, etc. as deemed necessary for meeting mission goals
- Life Cycle Costs include hardware, application development, long term support over the entire life cycle
- Delivered Performance includes the following measures: nominal delivered throughput rate for independent transactions, computation-rate on a single application, responsiveness to dynamic overloads, and real time response
- Quantitative Performance and Risk Assessment - include accuracy and sensitivity of quantitative measures that portray delivered performance, schedule and cost estimates
- Application Access - includes common applications, tools, languages, operating systems, standard open system interfaces

Users of computers want computers that solve their applications reliably and swiftly. Users do not care about the type of computer. They are concerned only that the system meets its criteria as derived from their mission statement. Benchmarks for parallel machines must include measures of integrity and delivered performance on similar applications.

6.3.1 Physical Constraints

Tools are lacking that measure the resource consumption of various choices as well as their performance potential, programmability and program suitability.

Additional factors arise in C^{3}I systems that complicate the analysis of the potential use of parallel computers. These include limitations on size, weight, volume, and power consumption. These constraints make it difficult to analyze the application, the system and the computation expansion characteristics of the application. The typical use of many parallel computers as compute engines for single large applications lends itself to being matched to C^{3}I system requirements. Embedded parallel processors are also traditional for some portions of a C^{3}I application. For example, processing data at its collection point to reduce very high data collection rates down to reasonable internal system communication rates.

6.3.2 Availability

Many of today's parallel hardware and operating systems are not suitable for high availability applications. C^{3}I tests of availability are based upon mission criteria that may be complex and difficult to create and evaluate on parallel computers.

Availability is a test that requires long test periods while running complex applications. These tests stress all the features of the system. In commercial application's users expect parallel processors to match their positive experiences on workstations, servers, supercomputers or mainframes. They expect hardware to be reliable and fault-tolerant. Their high availability expectations require that parallel systems allow replacement of failed modules without full operational loss. Similarly, they expect operating system and application software to be reliable and repeatable. C^3I systems have more extensive requirements for fault tolerance and recovery since human life and mission success may depend upon the availability of the processor. The present status is that most "massively" parallel machines meet research environment expectations, not commercial ones.

6.3.3 Delivered Performance

Performance disappointments continue. They are a result of failure of parallel architectures to provide effective execution on the full mix of application character types.

General performance delivered by a parallel machine depends upon several detail level factors. The way those factors change with changes in the number of processors, p(n) is a primary consideration. The factors and the primary influencing design issues are:

- The effective execution rate of the node computer on sequential code operating as a stand alone processor, cache memory, local memory, access to input/output at the node
- The effective node execution rate within the parallel system with no application intertask communications, i.e., the processor effectiveness during interconnect latency (for both multitasking and multithreading), the link bandwidth from the processor to the system, uniformity of access to input/output and operating system overheads
- Performance of the interconnect structure - capacity-scaling, small latencies (application intertask communications timing)

6.3.4 Programmability & Portability

Performance remains too brittle. Component character and architecture to time-response-goal mismatch often causes significant effectiveness reduction.

The von Neumann model gets acceptable performance for applications programmed with functional goals and no great concern for performance. This initial performance can often be improved using a mix of non portable methods. Some programmers take the extra effort and accept the long term maintenance disadvantages of non portability. They typically get a performance improvement of a small constant factor. Parallel computer programming should be no different. Programmers should obtain acceptable performance with good portability with a moderate first level of effort. They should expect that additional performance obtained though mapping would be non portable.

Portability is one measure of programmability. The industry problem of attracting the independent vendor into massively parallel computing is one symptom showing that programmability is below acceptable standards. Some programming models (e.g., PVM, Linda, etc.) attempt to execute on both networks-of-computers and on tightly-coupled computers. Such a broad attempt yields a program that cannot provide acceptable performance on either target machine class across a wide range of applications. An application design, which serves widely different machine types, compromises the programmer and requires extra effort to map to very large latencies for the network machine. This mapping reduces the concurrency that could have been expressed and executed in the tightly-coupled, low-latency machine. This reduced concurrency limits the performance potential on the low latency machine. Therefore, dealing with a wide range of latency for different parallel computer types is not a realistic goal. Advocates of portability techniques do not clearly define the domain of portability for their products. They should distinguish their product's suitability for both networks of computers and types of parallel computers.

Automatic compiler success is another measure of programmability of parallel computer architectures. An automatic parallel program generator that gives effective performance would be an important advance. However, recent experience shows that efficiencies are too low on many parallel machines for automatic program analysis. Present data decomposition, necessary to use high performance compilers, has lead to disappointing results unless the applications were limited to highly structured ones.

Evaluators of parallel software systems say that porting effort is inversely proportional to the latency of a parallel processor. Decreasing latency impact by scaling the application to larger size is often a solution for only part of the application. Engineers make careful designs of interconnection systems. They seek to reduce the time spent for message transfer into and passage though the interconnect. Their efforts are in vain if delays for communications protocols dominate the total delay. High latency causes the programmer to build applications with higher computation content than communications. This effort reduces the opportunity for parallelism and limits the application's potential for higher speed through parallel computing. The use of coprocessors to overlap the protocol latency with computational use of microprocessor has not been as successful as one would expect. This is most likely a reflection of a concentration on large-message passing as a model instead of packet-size message passing.

6.4 Relation of Needs to Capabilities

Successes on narrow application types have induced vendors of today's parallel computers to finely tune their products to a specific application's execution. When application character and time-response-goal go outside the narrow case, then the machine loses its effectiveness. Software engineers need an adequate foundation to resolve this dilemma.

Like military C^3I systems projects, large-scale, mission-critical decision support applications require excessive labor, maintenance and performance risk. A software engineering goal is to find a solution for this daunting problem. Solving complex decision applications by delivering quick results is the area that the industry can make the most gain. The commanders that make these decisions are willing to pay for solutions in their decision making process. They care not for the technology used, but are unwilling to change their needs to match the computer's shortcomings. A significant payoff would result from use of robust capability-scalable parallel computers. Interfaces to legacy systems are necessary. Such systems require a robust balance of I/O and computational capability. Software engineering practices are feasible for computers that deliver promised performance on portable programs with high complexity and dynamic action. Software engineering practices cannot solve their applications if their computers lack the capability to provide adequate support for mission critical applications.

6.4.1 Application Suitability Uncertainty

Another major issue in use of parallel computing is the uncertainty of application suitability and parallel performance benefits in the face of the complicated architectures and unknown application characteristics. Deciding when a complex application matches a parallel computer may be difficult. Today's machines have resulted from tuning to a different set of goals than that of most C^3I system builders.

6.4.2 Demand for the Most Difficult Application Character

Novel human computer interaction mechanisms are providing new opportunities for building systems that are dispersed in unique and different locations. Wireless terminals, power pens, voice interaction, point-of-operation terminals, etc. may provide significant opportunities for systems building that were not previously available. The interaction of humans with a parallel execution should be a goal since human guidance could lead to a more effective solution in both quality and time of execution. Interactive control requires an extreme of dynamic resource and computational scheduling. It also requires that input/output in multimedia forms are required to be integral parts of the computation.

6.4.3 Infrastructure Weakness

Some still expect that software solutions can be found to solving problems of inadequate bandwidth capacity growth in cost-scalable multicomputers. As long as mapping is required, extensive applications cannot be ported with dependable performance expectations. General source compatibility of software for general and arbitrary applications across different cost-scalable multicomputers is not feasible unless a common denominator is found. That usually will compromise the performance potential for all the multicomputers. The structure, latency and bandwidth capacities of multicomputers differ too drastically and are often inadequate for any but their selected application sets. Some cost-scalable multicomputers are simply inadequate for portability of applications, each one must be mapped carefully to obtain reasonable performance. Also, the research community often judges software tools by how well they meet the widely varying demands of a wide range of available machine architectures and sizes. Yet these multicomputers are not suitable for applications beyond their suitable applications. Such an expectation is unrealistic.

Both software tools and parallel hardware have become specialized for a set of applications that fail to represent the complex and dynamic executions characteristic of C^3I systems. The parallel industry has selected application types and as a result the process of evaluation of both tools and parallel computers has been distorted. Computers must be judged based on a broadly representative application set because the builders cannot anticipate the applications to which they will be put. By concentrating on an applications set that can be mapped to cost-scalable architectures, researchers have created non-representative methods, tools, and evaluation criteria for parallel computers. The result is an inverted rating scheme where software tools are expected to fill the missing capacity of multicomputers that are suited only a structured and stable applications set. Application selection cannot drive the hardware if the hardware is expected to fulfill a wider role than its design criteria. Annex A - Appendix C gives a brief overview of distributed computing software.

A key measure of a parallel system is the number and diversity of independent software vendor tools and applications. Only one vendor type has accepted parallel computers enthusiastically -- the data-base-manager application. The most common data-base-management systems are ported to a several types of parallel computer. This is due to the use of these machines as servers, performing the single parallel data base application for many clients on a network. Other applications remain on the workstation being fed their data from the servers. However, acceptance among other independent software vendors has been weak. They have a hard decision to make on today's parallel machines. The diverse number of architectures requires that a standard model be used for all before designers and programmers fine tune for machine dependencies.

6.4.4 Need for a Parallel Programming Model

We need a programming model for shared memory programming that allows symmetric multiprocessing code to be source-compatible with local memory based multiprocessor code. A common set of agreed upon standards that bridge between cost-scalable multicomputers and shared memory multiprocessors is necessary. A low latency version of the Nexus specification (Aerospace Corporation) is possibly an example of such a message passing system. If used with only small packets it may be suitable for both symmetric multiprocessor and local memory based multiprocessor standard for communications.

6.5 Expressing Parallelism

6.5.1 Parallel Machine Models

Programming models that apply only to the least difficult or restricted application characters are not adequate as a foundation for portability combined with effective performance. An acceptable software engineering programming model should allow programmers to create mechanisms for meeting the C^3I factors. Tools should be available that allow the designer to select suitable machines to use any application character and to match a time-response-goal.[6-2] A tool is needed to predict performance by evaluation of architectures against different application character mixed. The character step requires a model of computation that quantifies the characteristics. The application's character and time-response-goal are a necessary part of the evaluation. An engineer's evacuation of the suitability of and architecture requires a virtual model of parallel computers and a instruction set. The model and instruction set must be complete and sufficient to model both shared memory and capability-scalable computers. The performance prediction requires a level of robustness in the computers to avoid non-linear bandwidth effects. To complete the system feedback input is needed from several different system implementations.

Defining a set of application types that match the needs of C^3I subsystems is a useful exercise. The set is a qualifier for a parallel computer. The performance of those application types on the processor provides a measure of how the parallel computer might do on a future C^3I system. Such matching would require a wide range of application types to include the diverse needs of C^3I systems. The selection could include process control, transaction processing, data base, engineering simulation, economic modeling, etc. An important tool would be one that measures the concurrency, grain, complexity and parameters of these applications to allow comparison to the content of C^3I systems.

In order to make any quantitative approach to parallel software engineering the application character must be identified and related to known performance impacts at each stage of development. Measurement tools for all aspects of characterization would have to allow estimates of character, models of characters, benchmarks and measurements, and actual code of varying character as inputs. The effort is equivalent to rebuilding an electronic automated design tools set, the associated libraries of behavioral models, benchmarks, and timing estimation tools for parallel computers.

[6-2]Note the proposal of a split of responsibility. The machine architecture is responsibility for a foundation for creation of means of achieving C^3I factors and a time response type. The software engineering industry would be responsible for a programming model that was suitable for all application character types.

6.5.2 Parallel Exacerbation of Software Engineering

For conventional computing, software engineering processes have a factor-of-safety due to resource capacity and performance growth provided by the rapid progress of technology. However, parallel systems violate assumptions of conventional software engineering processes. Rapid obsolescence of parallel computers, long procurement cycles, the lack of a quantitative model for performance assessment effects these processes. Integrating quantitative methods into the parallel software development process is necessary for an ideal environment where each phase accurately reflects the expected performance of the system being built. Enhancing the spiral method for performance modeling at each phase of development for parallel computers is one approach. To accomplish this goal there is a need for an architecturally independent programming model that couples to quantitative performance models. Such a programming model either requires capacity-scalable interconnects for every target parallel machine or requires development of analysis techniques allowing compilers to deal with the nonlinear response of interconnects.

6.5.3 Conflicting Measures

The purpose of parallel computers is usually to provide higher speed. Speed can mean a higher throughput for independent transactions or it can measure the rate at which a large single application can be completed. Speed can mean response time to an external interrupt during execution of the single large application. Engineers use other measures of speed that match specific C^3I system needs.

An application with fine synchronization grain and high parallelism uniformity is one for which parallel instruction communication delays become insignificant. (Terms are defined in the characterization method in the application essay.) However, the process of creating applications with large grain is difficult. It requires selection of algorithms, mapping and often unresolvable decisions about dynamic portions of the application. Most often the degree of parallelism is reduced as the synchronization grain is increased. The uniformity of parallelism is aided by having many small grain tasks.

Note the conflict between the efficiency attained by large grain creation and scheduling improvement gained by high and uniform parallelism. With more tasks the parallelism is high. When programmers combine tasks to create large grain modules, they reduce the number of tasks available for easing scheduling inefficiency. The solution: increase the computation-to-communication ratio without decreasing the task number or affecting their size differences. The result is that many parallel processors are capable for selected applications where each task is identical at each node. Data is mapped to the nodes to keep communications low. This type of processing is called single-program, multiple-data (SPMD) and is typically the mode in which a cost-scalable multicomputer is used).

6.5.4 Assumptions About Programming Models

It is commonly believed that shared memory and distributed memory systems require locks/monitors and message passing respectively as programming models. This is not necessarily the case. The programming styles of shared memory systems with implied freedom from mapping and distributed memory message passing both have advantages. Shared memory systems provide a common name space but require concurrent access control by locks and monitors found by many to be difficult to program correctly. Many reliability problems result from the difficulty in correct and provable codes for shared memory systems. Message passing automatically provides protection and need no locks and monitors on the user's part for concurrency correctness. Message passing is conceptually less difficult to program safely. However, message passing s blamed for the mapping required to achieve good performance when there is a very high overhead for creating and delivering a message. Then, the programmer must create large messages from an application that naturally has many small ones. If messages were about as efficient as a shared memory access then the user related problems of message passing disappear. The system vendor would provide object naming and protection between each execution unit. This point of view points out that weak parallel systems capabilities (high latency and low bandwidth capability) have led to the wrong view of message passing.

6.5.5 Feasibility of Programming Models

Shared memory multiprocessing would be more widely used if vendors overcome reliability cost of locks and mutual exclusion operations. Similarly, with adequate bandwidth and low latency for delivery of small messages, vendors could create distributed memory systems that are effective without mapping. Distributed memory systems with hardware support for interface to the interconnect system can mimic shared memory. Both are viable and a common programming model is feasible on either if adequate hardware, compiler and operating system performance and features are present. The result is message passing protection between execution units within a common object name space.

6.5.6 Ideal Programming Model

The ideal programming model is a message-passing interface which hides the architecture's details. It must also provide its user with private memory safety of locally protected memory for each process. The hardware and operating system could be shared memory or distributed memory. A criteria for existence in the marketplace will be that the specific architecture would be invisible, in fact might change over a single manufacturers line. For example, AT&T's symmetric multiprocessors go up to 32 processors. Their distributed architecture goes up to hundreds with the same programming model. This interface should allow programmers to create and transmit both small synchronizing messages and large data transfer messages. Shared memory machines would provide the same small message, safe programming. Large scale systems should have to provide low latency interconnects for similar message sizes. On larger machines multithreading (switching in another task) would hide interconnect latency. Without hardware and support software that meets this model a common programming model is difficult.

6.5.7 Future Goals

One hopes that a parallel compute model will emerge and that systems vendors will then use the timings of its individual operations as the test of performance. Vendors could add the extra hardware necessary for some markets and leave it out for others, but engineers will make the addition and subtraction quantitatively. Thus, architecture and programming design will be separated.

Annex A - References

Parallel Algorithms for VLSI Computer-Aided Design, P. Banerjee, PTR Prentice Hall, Englewood NJ 07632

Computer Design, Special Report: CASE Tools for embedded and realtime applications, Tom Williams, Editor, April 1994 pg. 65-78.

Henry J. Bush, Presentation, Thrust 3 Program Overview, 21 June 1994

RADC-TR-89-337, Final Technical Report, January 1990, Rome Air Development Center, Griffiss AFB, NY, Software Techniques for non-Von Neumann Architecures, C. Lightfoot, et al., Computer Sciences Corporation

Electronic Engineering Times, August 1, 1994, Executive Viewpoint: The process is changing, P. Yeatman, Radstone Technology, Montvale, NJ.

IEEE Spectrum, May 1993, Focus Report: Workstations and PCs, M. Furtney and G. Taylor, Of workstations and supercomputers, pp 64-68.

F. Baskett and J. Hennessy, Microprocessors: From Desktops to Supercomputers, SCIENCE, vol 261, 13 August 1993, pp 864-871.

Computational Applications in Manufacturing, A. Formica, RCI Management White Paper, 1994.

Clustered Workstations: The Dominant Parallel Architecture?, J. Mohr, RCI Management White Paper, 1994.

Fundamentals and Practicalities of MPP, G. Astfalk, The Leading Edge, Part 1, August (p 839), Part 2, September (p 907), and Part 3October (p. 992) 1993.

A Spiral Model of Software Development and Enhancement, B. Bohem, Computer May, 1988.

A Model for Information Technology Adoption, C. Murphy, Accord Solutions, Inc. Tech Note 94-001-CM, 1994.

HPC Research Note, October 27, 1993 Gartner Group, H. Richmond, MPP Systems: A Strategic Weapon for Industry

High Performance Computing: The Potential Contribution to General Purpose Computing in the 1990s., Gartner Group, H. Richmond, 1993.

H. Richmond, "Commercial Applications of Parallel Processors," Austin, TX, July 1993.

"Software Engineering of Parallel Systems," Carl Murphy, Encyclopedia of Software Engineering, JWiley&Sons, 1994.

"Future Directions in Supercomputing," D. Robb, Supercomputing '93.

Annex A -- Parallel Software Engineering Assessment

Appendix A: Definitions & Relation to C^3I Systems

1. Networks of Computers

Networks of computers are multiple workstations organized in a loosely coupled local area or wide area network. The protocols used for information transfer between nodes are often based on telephonic requirements. Those protocols are efficient for large-size message passing. A dedicated network of homogeneous multiple microcomputer nodes is a Cluster or Farm.

Workstation clusters are standard workstation processors that connect to each other via a network, not an interconnect backplane. Two forms are available: those built from dedicated, rack-mounted workstation processors (without a display unit for each processor) and those that are networks of desktop workstations. We will refer to the first type as clusters and the second as "farms." Clusters and Farms operate effectively on replicated copies of applications executing the same code on different parameter sets. Clusters also can be dedicated systems that operate as throughput engines supporting multiple engineers, each running a separate copy of an application. Many engineering problems require repeated computation over different parameter sets. System administrators can set up farms to run overnight, also for replicated copies of applications applied to different parameter sets. The Farm approach has appeal to overnight use of "idle" workstations, thus optimizing an organization's use of compute resources.

2. Multicomputers

Distributed multicomputers are computers that theoretically "cost-scale." Adding a new module costs the same, irrespective of the number of modules already there. Cost-scaling computers have reduced bandwidth capacity as machine size grows. Each module adds a fixed bandwidth capacity so the capacity per processor decreases as the number of processors goes up. We call these architectures, "cost-scalable," because the addition of a module adds a fixed amount of bandwidth capacity to the system, theoretically at a fixed cost per module. Therefore, cost is directly proportional to the number of processors. In theory, by doubling the size, one doubles the price.

There are specialized applications where the multicomputer hardware is more cost effective because the application matches the processor. Increasing the size of an application typically increases its communication demands proportionally. An application can only scale in size on these machines if its portion of communication per size goes down as the application's size increases. Some algorithms achieve this and are useful in research. However, the engineering process requires a complete solution for an entire application and such algorithms are elusive. In addition, the "hardware-only" cost evaluation model is inaccurate if software and integration development is necessary to map production applications to the hardware. Evaluations should include the following: expected costs of the operating control software and application software, the expected effective performance on the projected application, and the potential for obsolescence of mapped applications.

3. Multiprocessors

Multiprocessors have increased bandwidth capacity per processor module added. This gives approximately constant bandwidth capacity between any arbitrary processor for any size machine. They have an interconnection that provides increased bandwidth capacity per processor module. This increase is that necessary for approximately constant bandwidth capacity between any arbitrary processor pair for any size machine. Increasing the count requires paying the costs of extra bandwidth necessary for maintaining the fixed bandwidth between arbitrary pairs. In theory, multiprocessors should be capacity-scalable. Multiprocessors include either a local-memory based or a directory-based shared memory based multiprocessors.

With local memory multiprocessors that are capacity-scalable, applications may increase in size and communications as the processor size increases. This allows a single algorithm for any size application or machine. However, those machines grow in cost since added bandwidth capacity per processor maintains the balance between application demand and processor size. The tradeoff of hardware cost against system software development and software maintenance over a long life is, however, seldom applied. Considering the cost of development of operating systems and tools, they compare more favorably with the costs of multicomputers. Including the costs of software for system integration, application development, and maintenance the multiprocessor often is more cost effective than a multicomputer.

4. Symmetric Multiprocessors

Symmetric multiprocessors (SMP), are multiprocessors that share a common memory. They typically have an upper limit based on the common bus used to reach the shared memory. The programming model assumes consistent shared access to memory from any processor. Obtaining good performance on these machines requires good locality-of-reference. Cache consistency mechanisms control the bus access hardware to keep cached shared values consistent. This small processor count type is important in the marketplace. It offers a smooth path up from the workstation. The symmetric multiprocessor is now popular in commercial client-server environments. It is displacing mainframes in some important multiuser information and technical applications.

Symmetric multiprocessors typically use a fixed resource bus and are not capacity scalable multiprocessors. Therefore, changing the number of processors changes each node's share of the fixed bandwidth capacity and increases the latency to reach memory due to bus contention. However, within those limits, the operation is less brittle than on distributed multicomputers. Programming is very portable since there is no mapping necessary. However, shared memory programming uses semaphore and mutual exclusion operations to coordinate memory access. Those approaches are error prone and difficult to program.

5. Multiserver Systems

Workstations and their graphical user interfaces have brought the user closer to the computation. Putting the user at the display has significant advantages. The entire process of application set up, computing, and results analysis works directly with the user's immediate environment. However, if the computational period of the engineering process is too long, the close coupling of the user to the application is lost. Keeping this coupling requires a link to an external computational engine that can deliver quick results. Often the files and applications are common to several users. The user uses them from external files or data bases. Multiple users require access at high transfer rates to use and execute as a team. High performance local networks now provide this capability, providing data, graphical display and computational, and data base capabilities. The introduction of very high performance fiber channels to this mix will allow more rapid client-to-server transmission.

A C^3I system is typically a multiserver one. Each component may have a different time response and application character as necessary to meet their mission. Often different developers build different components. Parallel processors often serve in these multiserver networks. Architects tune them to meet their individual component's mission. This assessment focuses on closely coupled parallel computers because of the need to control the response time of the application being executed on the parallel computer.

6. Heterogeneous Systems

We define heterogeneous computation as breaking a computational intensive component apart and performing each part on different computers. [1] The goal of heterogeneous computing is to separate out a communication constant part of an execution-bound component. As the application size changes, the new part's communication remains constant while the remaining part's communication expands. If the surface between the parts has constant communication intensity with application expansion, then the communication constant portion can scale on cost-scalable computers. The remaining part requires another specialized or capability-scalable computer.

Note that the introduction of heterogeneous solution to a component reflects back to the general system design. It may mean that the general system design could have recognized the interface(s) created by the heterogeneous split of the component. The chances of success of the heterogeneous computing approach depend upon the application. Extremely high communication rates between mixed processor types are necessary for execution-bound computations. On each iteration, the system must reformat and pass all surface-related information. In addition, this method may lose the advantages of data-locality-of-reference in caches or local memories. The integration, development, programming and life cycle expense for multiple parallel computer types must be weighted against the extra expense of building a more general purpose parallel computer.

[1] A C^3I system design allocation breaks a system into different components, each with its own mission. These likely have different time resource goals. The transfer of information between the components is part of a system-interconnect-component's time and cost budget. From this viewpoint, a C^3I system is a multiserver system. Within each component there are often parts that are execution bound. We call any splitting of an execution bound part into additional mixed server communicating parts, "heterogeneous." Any additional storage or communication delays or resource use are added costs of the time and resource budget of the split component and not the interconnect component.

Appendix B: Status of Softare Engineering

1 Workstation Capability

Workstation based system design and development processes and tools are now maturing. Applying them carefully provides a project team with a well-managed path to system development. The process of software engineering has evolved to provide a basis for tools and for managing the building of long-lived, complex C^3I systems. Steady improvement by tool builders has provided a steadily improving progression of process environment tools for system development. The results are Computer-aided-software-engineering (CASE) environments, built-in configuration management (SCCS), and reusable language features (from Ada and Object Oriented languages). Personal computer programming systems now feature some encapsulation and code reuse aspects of modularity and libraries. For example, Microsoft provides VisualBasic data and code encapsulation "objects." Their Dynamically Linked Libraries aid programmer productivity and maintainability. Workstation vendors provide similar capabilitese for UNIX systems.

2 GUI Created Demand

The successes of graphical user interfaces let the user request more complex and resource demanding applications. These, in turn, create a new demand for higher performance and capability. A paradigm of close user interaction results. The resulting cycle is setup, computation, and analysis with each step involving intimate user control. Users want complete control over the execution without the interference of detailed coding. Often the developer's dilemma is that user expectations grow faster than performance. The productive cycle of interaction breaks down if one part of the interaction takes too long. Thus the need and requirement for acceleration of the slowest component.

3 Holistic Interactive Operation

Technology influences those processes. This rapid change gives the user increased expectations during system development. Since new features become feasible as technology advances users expectations grow. Command capability provided by Graphical User Interfaces(GUI), specially the object oriented ones, let the user increase demands for features and performance. The computer industry has reached a state that users expect a holistic problem solution environment. This is where users interact with the system in short turnaround times for an entire solution process of preparation, computation, and analysis. (For example, VisiCalc and its modern descendent spreadsheets provide this create, compute, observe, and repeat.)

A "holistic" approach by human interaction to reach "non-canned" solutions is a leading example of growing expectations of processing responsiveness. Corporate reengineering projects show that information systems give a "case worker" the capability to do an entire process. This approach is far more efficient than using several specialized workers. Instead this new model of processing is that the user gets results in adequate time to iterate the solution several times in a day. Thus, demand grows due to increased human interaction with the computer system. This demand exists in many professional projects: business decision making, software development, mission planning, computer-aided-engineering, and control systems.

4 Computer-Aided-Software-Engineering (CASE)

For sequential computing, CASE tool vendors define the phases and processes in the software engineering process in their tools. Figure 6 - 1 gives one vendor's view of standard components. When used to complete each cycle of the Spiral Method, a CASE system is a useful tool for management of the development process. The CASE process shown has a critical assumption: that the hardware and software of the system can be separated by interface constraints. The CASE process assumes that the hardware will be available for execution of the software when the testing phase begins. System design analysis is supposed to establish that the hardware system meets performance requirements. The complexity of interconnect structures makes this a formidable task. Lightly stressed interconnect structures require less demanding analysis. Otherwise, the latency and demand constraints are nonlinear. Physical constraints in C^3I systems sometimes limit the designer to reducing the interconnect capability until interconnects are well used. Standard processes and CASE tools do not provide the necessary interaction mechanisms between system hardware and software design to accomplish this task. C^3I applications are too complex and dynamic for tools that require one to build large grain modules. In von Neumann architectures, the rapid advances in performance and capacity are powerful compensators for these errors. CASE developed systems often succeed due to the factor-of-safety provided by technology advances. One reason for the spiral method's success is that hardware design and software designs merge along with the user and developer's perceptions of the system.

The insertion of parallel technology negatively affects the validity of the CASE process and its assumptions. The interaction of interconnects within a parallel system is not suitable for closed form analysis for applications of the complexity that require CASE tools. Software developers must consider the details of the hardware organization and bandwidth capacity along with the software effort. Setting constraints (or goals) for software modules is no longer a suitable method of separating software and hardware. To correct this assumption, CASE systems need to allow a similar process to integrated circuit development tools. The electronic design automation process offers a model for these changes. Tools for electronic design automation provide a paradigm useful for the software engineering process for parallel systems. Figure 6 - 2 shows that process.

5 Comparison of CASE and EDA

Electronic-automated-design (EDA) process tools use system interaction models as an integral part of each design phase. Languages that support the EDA process allow design expression at all levels: behavioral, register-transfer, standard cell, gate, and transistor level design expression. At each stage simulators execute the design and provide functional tests and quantitative timing and resource responses. At each stage, the designer defines the organization, placement and interconnection of the components. Mapping to a specific foundry occurs at the gate-to-transistor interface, allowing portability of the design to various sources. Foundry-rule libraries are the analog of the software developer's use of extensions to standards. Thus gate-to-transistor mapping is based on a standard set of gate constructs that are the EDA user's equivalent of a virtual machine. Engineers use automatic and hand guided tools in the last stage to match foundry design rules. Object oriented methods are commonly used throughout the process by vendors of EDA tools. During each transition between phases, engineers generate a set of tests and test vectors. Those tests provide a decision basis for passing to the next phase. Simulators of the actual operating environment are often used to place the design under expected stress. Test vectors for a computer chip come from running code of the type expected. For example, RISC computer designers use SPECmarks as the test generator. SPECmarks are performance indicators but are also used as test vector generators. Millions of lines of code run through the simulator to ensure that a processor chip is adequate for foundry insertion. Foundries guarantee that the chip produced will work (function, timing, and power) according to the test vectors used to accept the chip for production. *In both EDA and Spiral/CASE, the process advantage is the early resolution of the imprecision of initial specification.* The EDA system has the advantage that the target instruction set(gates) is significantly more restricted and well defined than that of the CASE system. EDA tool vendors are now in the process of reengineering their tools because the wire delay time can be ignored in feature sizes above 1 micron. Now that gate delays are not the dominant delay the accuracy of predicting timing delays across a circuit becomes less precise. The separation of gate delay from layout delay provided a hidden layer in prior versions. Their debate is whether the logic designer should learn layout methods and penetrate the layer in their next versions to get more repeatable timing delay predictions. The problem appears to be similar to that of "mapping" to architectures. Separation of hardware and software development is important to the engineering of parallel hardware and parallel software. Without this capability each effort by hardware vendors and software developers is ad hoc and has little potential for large markets necessary for commercial advantage.

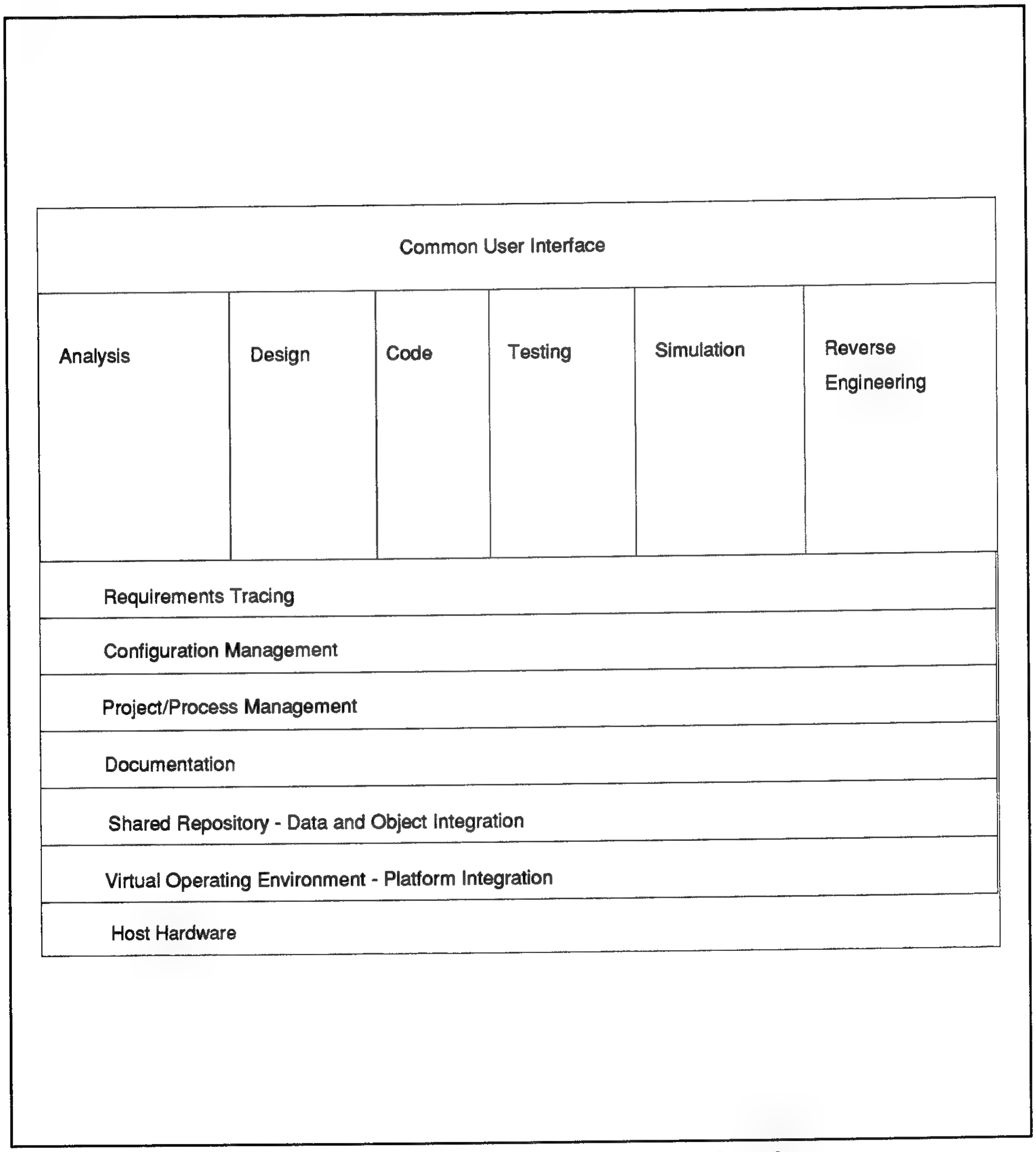


Figure B - 1 Computer Aided Software Engineering

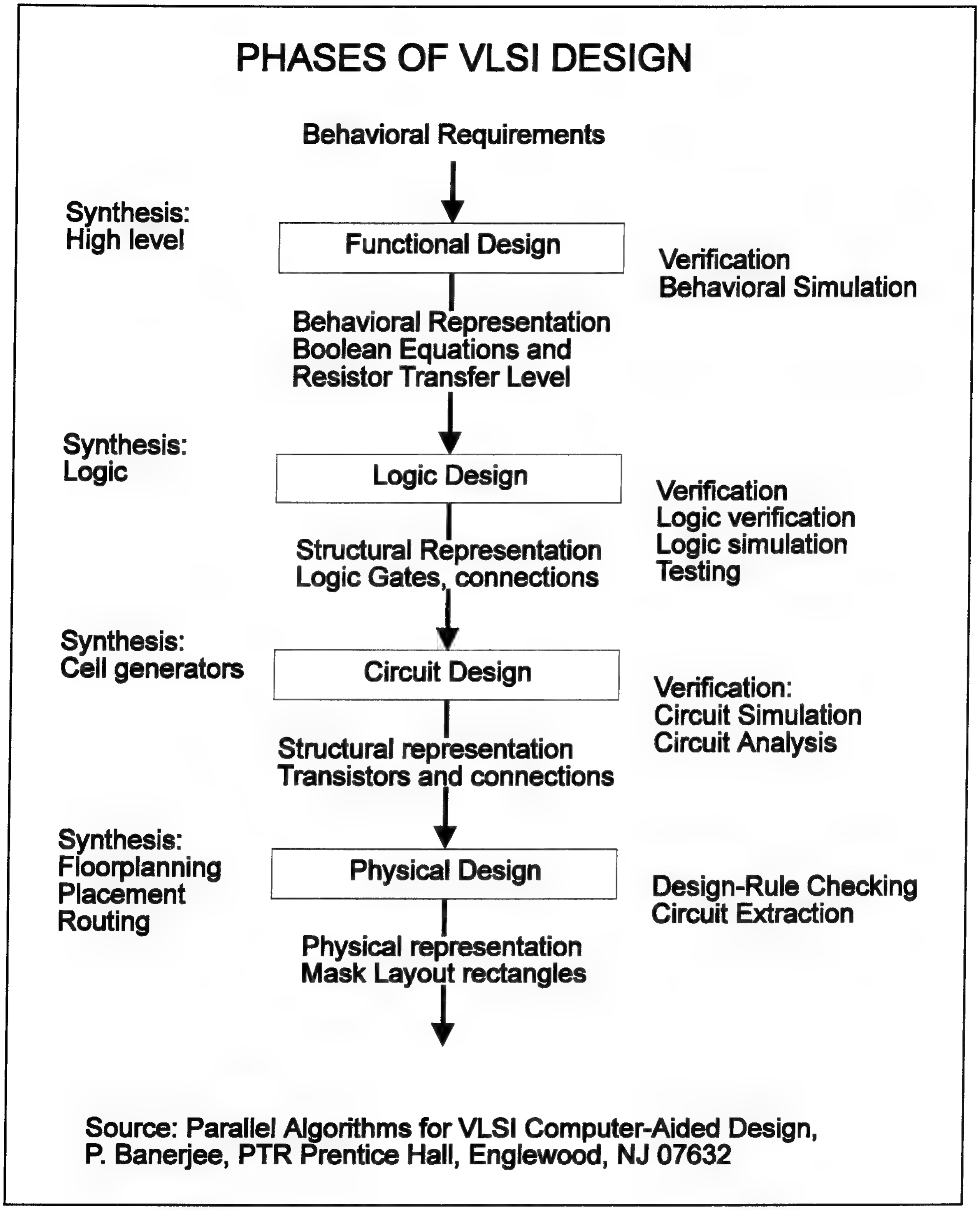

PHASES OF VLSI DESIGN
Behavioral Requirements
Synthesis: High level
Functional Design
Verification Behavioral Simulation
Behavioral Representation Boolean Equations and Resistor Transfer Level
Synthesis: Logic
Logic Design
Verification Logic verification Logic simulation Testing
Structural Representation Logic Gates, connections
Synthesis: Cell generators
Circuit Design
Verification: Circuit Simulation Circuit Analysis
Structural representation Transistors and connections
Synthesis: Floorplanning Placement Routing
Physical Design
Design-Rule Checking Circuit Extraction
Physical representation Mask Layout rectangles
Source: Parallel Algorithms for VLSI Computer-Aided Design, P. Banerjee, PTR Prentice Hall, Englewood, NJ 07632

Figure B - 2 VLSI CAD Process - Quantitative Results at Each Stage

Appendix C: Distributed Programming

1 Cluster Programming Methods

Message passing is necessary in clusters and farms. The protocol latency in clusters and farms requires application selection or mapping to get a high grain. This mapping process gives message passing a poor and undeserved reputation for difficulty and poor performance results. However, the difficulty in programming is due to the necessity for masking high protocol processing and not to the message passing paradigm itself. Languages for clusters and farms provide a message-passing interface but leave the mapping to the user programmer. Parallel computer vendors now commonly provide Parallel Virtual Machine (PVM) as a message-passing interface for clusters. PVM is likely the most widely used. PVM is widely ported and now cperates on the following:

- capability-scalable multiprocessors(e.g., IBM's SP2)
- cost-scalable based multicompters (e.g., Intel's Paragon)
- shared memory multiprocessors (e.g., Sun Microsystems servers)
- clusters
- networks of computers

This does not mean that applications that run well on one type will run well on another. Consequently, PVM ports provide functionality moves between architectures but not performance effective ones. For example, the mapping necessary to get good performance on a network of computers may reduce the processor count and speedup on the more tightly coupled multiprocessors. Similarly applications running on a large processor count IBM SP2 or Cray Research T3D would take advantage of the low latency. These applications would run poorly on any of the loosely coupled parallel machine types. Applications that do not maintain the level of grain (latency-ratio) expected at the network of computers mode cannot move across architectures and obtain good performance.

2 Parallel High Level Language

Researchers in high level languages for clusters and networks now concentrate on accomplishing general usefulness for arbitrary and multigrain applications without user programmer mapping. So far this is not a reachable goal. Products from Scientific Computing Associate's (Piranha), Parasoft (Express), and Meiko (Computing Surface) provide programming tools for replicated applications. There are also wide area programming tools, such as, ISIS, TaskBroker and Cronus. Oak Ridge National Laboratory distributes and supports the Parallel Virtual Machine (PVM) tool. NQS (EXEC) is a tool for programming of high throughput applications. A drawback of the cluster is the significant cost of software for replicated applications since each station requires a license.

3 Multiserver Programming Tools

Tools and mechanisms are also available for clusters of workstations and super large grain multiserver computing. These methods, protocols, networks, etc. apply directly to C^3I systems. For conservative uses, there is an established software engineering technology for dealing with clusters and large grain-size, multiserver systems. Products such as ISIS, Cronus, Linda, PVM, etc. can provide the necessary mechanisms for spreading multiple copies of applications across and managing the resources and processing among clusters.

4 Distributed Multicomputers

Distributed cost-scalable multicomputers are more closely connected than the network computers and less so than the capacity-scalable multiprocessor. Their vendors call cost-scalable ones "scalable parallel processors (SPP)." Their rational is that the addition of a module of processing power costs the same no manner the size of the machine. This essay calls that form of scaling cost-scalable. However, since bandwidth capacity does not grow with processor count, an application that runs on one size machine does not necessarily run well on another. To run effectively on any size the computer must have a property of reduced bandwidth demand per computation as the application size grows. Applications that fit this model are "data parallel" since data partitioning reduces communications significantly. There are several components of applications that have been found that fit this model. To make applications feasible on these machines they must be data partitioned (mapped) to minimize use of bandwidth capacity. Technology influences this approach by allowing the bandwidth capacity to grow in proportion to the computational demand. However, this growth must be faster than the speed growth of the microprocessors. Each of these changes affects the mapping of an application to the processor. Code portability is assured for only pure data-parallel applications. When communications are present, the nodes share the system's bandwidth capacity. Performance often falls off rapidly and differently on different size machines.

Software programming tools for cost-scalable multicomputers are Express, PARMACS, Linda, and PVM. This same tool set is also used on networks-of-computers, which have very high protocol latencies. Applications that perform well on networks-of-computers would be expected to do well on multicomputers due to a reduction in latency. Since only independent and very large grain (infrequent communicating) tasks can be programmed on networks-of-computers these are portable only in the direction from clusters to cost-scalable multicomputers and not in the other direction. However, expectations can vary. A cost-scalable multicomputer is often expected to provide a more dependable execution time since the interconnect structure is in a dedicated backplane. The network-of-computers expectation may well be only to keep idle workstations busy on an application during evening hours. In one case wall clock time is important, someone is waiting for the result, in the other only for completion by morning.

Appendix D: Models for Expressing Parallelism

1. Parallel Software Processes and Methods

1.1 Incremental Delivery Development Process

The staged development process is needed. This is a process similar to the VLSI design process (See Appendix B.) where each synthesis stage is verified and tested for functionality and timing before the next stage is synthesized. A component based system design would have the execution characteristics of each component defined and matched to the chosen parallel processor. At the next stage a detailed module program using only the generic machine instruction set would be used to write the module. Simulation and analysis would again verify performance goals. The last stage is the one in which the target machine knowledge is used to increase performance. (This matches the VLSI design rule stage where foundry information is first imposed.) This approach would be criticized by parallel advocates of many of today's machines because the effort of the last stage is significant. The value in the approach is that machines whose instruction set closely matched the output of the third stage would be more easily programmed. Those machines could be expected to cost more since additional hardware capability is required to allow them to perform that instruction set well. However, the experience to data has shown that the investment in operating system software and the attraction of independent software vendors favor the economics of parallel machines that provide availability and robust interconnect structure capability.

1.2 Rational Engineering

A rational engineering process is to develop the system in a series of incremental builds, each on a fast development cycle. Within each build, development managers iterate the design in stages to precisely define requirements, environment, and performance. After each build installation, testers evaluate the product to ensure that, as fielded, it matches actual requirements. The Spiral Method is one example [Boehm]. The result more accurately meets users' needs than a one-shot, large scale development. Tools for providing performance estimates for system designs (e.g., SES Objectbench and SES Workbench) are now available. These tools let the designer apply them for both hardware and software at very detailed levels. Combining the rational engineering process with these performance prediction tools significantly reduces risk. Developers need the same rational engineering capability for parallel components within the system.

2. Graphical Programming

The requirement for graphical programming is critical to the wide spread acceptance of parallel systems. Graphical system users can be relieved of detailed knowledge of both the parallel architecture and programming language details. Such users are necessarily high-level users that are building application by integrating lower level constructs. Programming using graphical methods are template and button bar approaches, where the parallel language commands and structures are available for integration into a working application by the user. The system requests the user for the logic for cases, loops, by entering it into a window which guides them through any language specific logic expression details. The system can enforce good programming practices by querying the user for the action to be taken for exceptions, etc.

The goal of programming should be to provide a very high degree of reusable modules. For a project the engineering of the detailed modules would be left to a small set of the most power fulprogrammers. Others integrate the detailed objects into the specific application. However, graphical programming must express the "problem architecture." Graphical mapping tools intended to match an application to a particular parallel processor defeat the portability virtue that should be available. They are often applied at the wrong level. Only power users should be doing the detailed mapping required to map the common button bar commands to a particular processor. Those commands are a virtual machine programming model. The system displays template and button bar calls on the graphical screen. Only in this fashion can users and vendors gain the benefits of porting a small command set.

3. Debugging and Performance Tuning

A separate element is the need to provide correctness debugging and performance tuning. These tools provide instrumentation and constraint probes of the parallel application to control the operation to verify correctness and to eliminate performance bottlenecks. Debugging and performance tools should have a proven interface to selected operating system(s). They should also have an interface to the parallel system's "run time executive" that executes the application'sparallel operations and parallel intertask commands. Measures should be the following:

- individual time delays for each of the parallel commands provided as experienced in the application
- time delays for the average mixture of parallel commands of the application
- concurrency - count of the average number of ready tasks
- instantaneous concurrency - count of ready tasks during a step
- slackness - the concurrency minus the number of processors
- granularity - the ratio of normal instructions to parallel instructions
- potentiality - count of waiting instructions (total and by value)
- kernel accesses - count of access to kernel interface
- external accesses - count of accesses to operating system
- context switch count - the number of processor swaps (by application, kernel, or operating system)

3.1 Performance Tools

Performance tools should provide a postmortem or analysis after completion of application. They should provide the information necessary for performance prediction - efficiency and speedup equations - based on the measured timings of the parallel instructions and their mixture within the application. The tools should make a recommendation for processor count based on concurrency and provide a processor architecture timing data base.

3.2 Correctness Tools

Correctness tools should interface to a commercial debugger that is commonly used in normal workstation programming. There should be history files created during debugging and constraints on execution that allow repeatable (deterministic) operations when active. This control requires control of repeated program execution between set break points, or between barriers, or between identified steps in the execution. A history of communication events is a necessary element of the data base that controls these operations.

3.3 Information Visualization Tools

Parallel programmers must manage enormous quantities of information and complex program behavior. To understand these operations visual aids are one solution. To be effective, they must provide a proven connection from graphics programming to object-oriented languages. Communication displays should allow programmers to evaluate correctness. The language should allow programmers' expression of complex interactions built up from primitive ones. The graphical system shoulddisplay execution graphs between pairs of barriers created to investigate a correctness error or a performance bottleneck. Tools should allow easy and structured building of a concurrent application from a small set of objects defined to execute the application.

4. Object Oriented Languages and Methods

On production environments in industry, most programmers create their applications by tying together shrink-wrapped applications bought from independent software vendors. They develop few applications from scratch. Independent software vendors consider portability, effective performance and reusability as important features. Object- oriented software engineering methods and languages are critical to their success. In addition, the applications are likely to be more complex and dynamic than present MPP methods allow. Therefore, parallel technology should provide high performance on applications expressed in C++ or other object oriented languages. Systems may store objects in object-oriented data bases.

Programmers need power languages to express complex operations that are difficult to express in standard languages or graphical programming. They require expressiveness for data partitioning and process interaction structures. Such languages must provide execution of the power expression and provide a performance prediction of the expression on a chosen machine. A good example, is Proteus, from the University of North Carolina. Many commercial firms require even simpler languages that are more easily learned. The corporation's information infrastructure may only have COBOL and Fortran experience. Smalltalk and other visual object level programming are important language choice.

ANNEX B

Rome Laboratory Blue Ribbon Forecast Panel on C^3I Parallel Software Engineering

Working Session and Position Summaries

1. Introduction

RCI, Ltd prepared and distributed an announcement and call for papers to obtain diverse input for the forecast panel to consider. Position makers were then selected to present their papers at the Forecast Panel's working session and to participate in the initial discussion of the issues of Parallel Software Engineering for C^3I systems. Panel members also stated their positions through papers and presentations. This Annex provides a short overview of the position papers and presentations. They are organized in the following hierarchy:

1.1 Introduction -- methods and goals of the working session and the forecast panel

- Joe Cavano, Rome Laboratory, "The Forecast Process"
  Carl Murphy, for RCI, Ltd, "Trends, Vision, Strategy and Forecast"

1.2 C^3I -- specific positions on C^3I system engineering

- Walter Beam, Independent, "Future of C^3I systems"
  Walter Beam, Independent, "C^3I data server workhorses"
  Robert Wasilausky, NRaD, "Parallelism through evolution not revolution"

1.3 Industry Level -- broad industry directions or viewpoints

- Gordon Bell and Jim Gray, Independent, "Scalable Computing 2000: Network & Nodes"
  George Lindamood, Washington State, "Parallel Technology for Everyman"
  Peter Seigel, Cornell Theory Center, "Globally Scalable Architectures"
  Jeffery Mohr, Information Technology Solutions, "Parallel Processing for Commercial Applications"
  Jon Webb, CMU, "The future of parallel software engineering in an economic context"
  Kim Gibson, Motorola, "Mobile Network Requirements"

1.4 System Engineering -- software engineering of systems with parallel systems

- J.C. Browne, Scientific and Engineering Software, "Systems Engineering of High Performance Parallel Computing Software Systems"
  Armand ten Dam, TNO Institute of Applied Physics, "Developing parallel applications: the Hamlet approach"

- Masami Uemoto, Concurrent Technologies Corp, “Future Manufacturing Engineering Environments”
  Karsten Schwan, and S. Yalamanchili, Georgia Institute of Technology, “Software Tools for Dynamic, Parallel/Distributed Real-Time Applications”
  Carl Murphy, Accord Solutions Inc, “A Parallel Strategic Symbolic Decision Capability”
  Jay Eckard, Oracle, “Enabling Technologies”

1.5 Programming Models -- design level engineering of applications

- Douglas Smith, Kestrel Institute, “Knowledge-based support for parallel software engineering”
  Jan Prins, University of North Carolina, “A refinement-based approach to parallel software development and maintenance”
  Stephen Yau, Arizona State University and Doo-Hwan Bae, University of Florida, “Architecture-Independent Object Oriented Software Development for Parallel Software Systems”
  Thomas Cheatham, Harvard University, “Bulk Synchronous Parallel Computing -- A Paradigm for Transportable Software”
  Rajive Bagrodia, UCLA and Mani Chandy, CalTech, “Definition of a Virtual Machine for C^3I Applications”

1.6 Programming Tools -- implementation level engineering of applications

- Andrew Sherman and Beverly Thalberg, Scientific Computing Associates, Inc, “Virtual Shared Object Memories: A shared data foundation for parallel software engineering”
  Craig Lund, Mercury Computer Systems, Inc, “Component programming and standards”
  David Rich, BBN Systems and Technologies, “Parallel software development: future of parallel and distributed software development tools”

1.7 System Effectiveness -- application or machine-specific supporting tools and infrastructure

- Min-You Wu, State University of New York at Buffalo, “Parallel software must support efficient execution of general applications”
  Martin Davis, Systran Corp, “ADAM: Application Driven Architecture Methodology”
  Sanjay Bhansali, and C.S. Raghavendra, Washington State University, “Tools for Parallel Software Technology”
  Matt Rosing, Pacific Northwest Laboratory, “Flexible and efficient abstractions for parallel programs”
  Robert Rabb and Lee Higbie, Seki Systems Company, “Issues in the Specification and Design of Parallel Programs”

2. Overview of the Position Process

2.1 The Forecast Process

Joe Cavano, Rome Laboratory, presented the forecast process. This is depicted by the flow diagram given in Figure B-1.

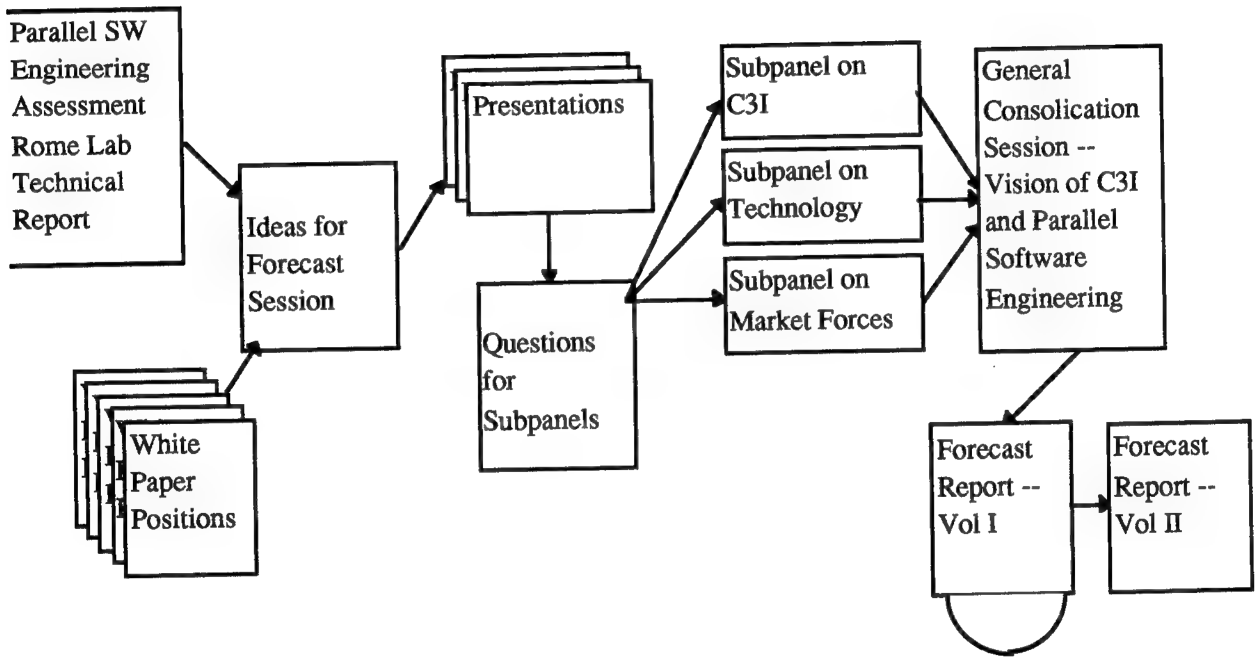

Parallel SW Engineering Assessment Rome Lab Technical Report
Ideas for Forecast Session
White Paper Positions
Presentations
Questions for Subpanels
Subpanel on C3I
Subpanel on Technology
Subpanel on Market Forces
General Consolication Session -- Vision of C3I and Parallel Software Engineering
Forecast Report -- Vol I
Forecast Report -- Vol II

Figure B-1. The Blue Ribbon Panel Process

Cavano explained that the diversity of ideas from the position papers would create a rich set of concepts for the forecast panel to include in their vision of parallel software engineering for parallel systems. The forecast report Volume I was to be prepared by RCI, Ltd and returned shortly after the completion of the meeting. Volume II was to report in more detail and include this Annex summarizing the position papers and presentations.

2.2 Overview of the Parallel Software Engineering Assessment

Carl Murphy gave the following paragraph to summarize the assessment:

The assessment's approach is to define C³I via a time-response goal and application criteria. These define an application's character profile. We used a time-response goal, frequency of synchronization, complexity, resource demand (or path) stability, and interaction mechanism to define the character of a C³I application. Determination of an application's character might allow a designer to estimate the effectiveness and feasibility of applying parallel computers to C³I system components. (Assuming similar character applications had been found to operate effectively there.) C³I components have a widely varying requirement for execution completion time. This varies in length and guarantee of repeatability. Based on that character scheme, we look at the present successful applications areas and evaluate how well they serve the mix of characters expected in C³I systems. We find that C³I systems are too complex, too dynamic, too real-time constrained and need to be too reliable to make good use of scientific research computers for all their needs. Today's parallel systems also fail to meet many needs of command and control systems, real-time control systems, symbolic decision making, and other time-response critical applications. The capability of today's parallel industry for C³I applications appears to be limited to a narrow range. Instead, a general parallel capability is needed to meet all C³I character types. In addition, the limited C³I suitability of large versions of these machines makes system building and life-cycle support expensive and difficult. There are too few systems or software engineering mechanisms that let designers design-to-deadline, predict reliable project budgets, schedule and estimate performance. Figure B-2 shows the Application Character chart used to describe C³I parallel system software. C³I applications tend to be in or nearby Position [2,2] .

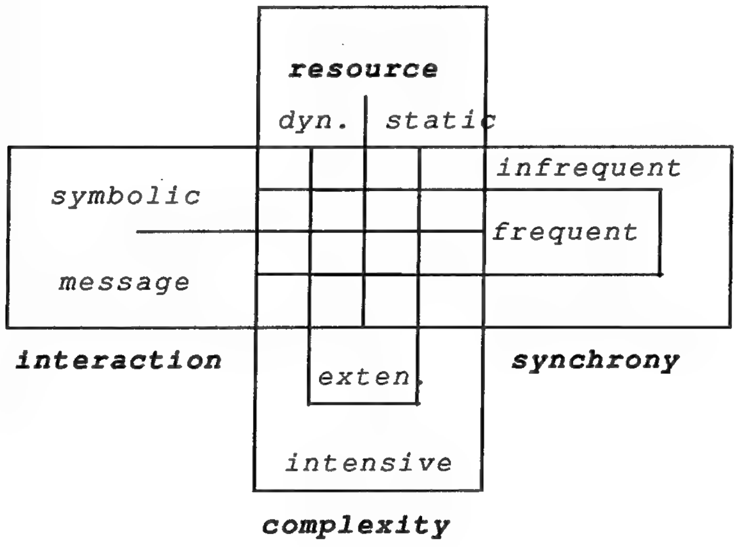


Figure B-2. C³I Application Character Chart

3. C^3I System Engineering Viewpoints

3.1 Future of C^3I Systems

Walter Beam reviewed the C^3I components given below:

- tactical communications/covert operations
- surveillance/radar/side-looking radar/intelligence
- target location and identification
- identification friend, foe or neutral (IFFN)
- force control
- force assessment
- tactical intelligence and analysis
- electronic warfare and countermeasures
- command, control, communications countermeasures and counter-countermeasures
- radio location
- electronic intelligence and analysis
- communication intelligence and analysis

The current factors given by Beam are critical to the panel's consideration in reaching a vision. Although recognized as a force multiplier the service leaders prefer weapons. Air Force locations vary significantly in global, aircraft type, positioning, and mobility. The many changes caused by the end of the Soviet Union demonstrate the need for highly flexible systems. The current factors are:

- downsizing
- local conflicts
- terrorists
- interoperability
- wide variety in enemy systems -- including US made
- grossly altered strategic situation
- limitations of technical intelligence
- national priorities
- overseas basing

The critical military issues that are important to parallel computers in C^3I applications are:

- availability and use of timely high resolution imagery
- rapid, accurate identification of IFFN air, sea and land combatants
- shortening cycle for target detection, identification, attack, damage assessment and recovery

- cooperative operations with Allies and other US Forces
- netting and fusion of target data from a variety of sources and broad-scale intelligence data

Beam identified the following process intensive C^3I applications:

- processing, storage, and retrieval of tactical intelligence data and imagery
- advanced multifunction displays
- speech processing
- sensor signal transforms (radar signal processing is the only extensive use of parallel to date)
- information analysis (netting, change detection, and fusion)
- future modes of use are not predictable
- exercise support by C^3I systems must expand

3.2 C^3I data server workhorses

Walter Beam pointed out that shared memory multiprocessors are penetrating microprocessor practice in much the same manner as they did in mainframe practice 20 years ago. Massively parallel processors are essential for only a few types of dedicated systems. These appear to have little future. The government will continue to have interest for image data transformation and analysis. He indicates that based on today's highly parallel machines the more important emerging areas are situation determination, display and simulation.

Tools for debugging and verification ensure that the results are independent of the machine state at entry. Personal computers, workstations and data servers can perform most C^3I activity. The common programming mode will be Visual Basic for graphical user interfaces. Executive controls multitask parallel tasks at run-time. The compiler sets them up in advance as separate tasks.

3.3 Parallelism through evolution, not revolution

Robert Wasilausky, from the Naval Command Control and Ocean Surveillance Center, discussed the refocus of the role of DOD computer research for defense specific purposes. Wasilausky identified the following areas of increased importance:

- human computer interaction
- automatic searching and information organization and presentation
- changing data to timely information
- automatic image recognition and access to large volumes of image and map data
- information warfare -- protection of our own and disruption of other's information systems
- digital library for command and control

Wasilausky pointed out that Navy Command and Control depends totally on commercial workstation microprocessor and network technology except in those special situations of size, weight, power, real time and computational speed demand. They use application programming interface standards and large buys to create economy and protection from obsolescence. He expects inexpensive commercial technology to be available for the following within the next five years:

- kernel-based operating system for operating system portability
- object-based operating systems for distributed, parallel processing
- PC operating system with Unix-like capabilities will dominate all platforms
- applications will become modular and independent of specific vendor technology
- multiprocessor workstations will become the norm
- clustered multiprocessor workstations connected via ATM will replace mainframe processors
- Real-Time system requirements will be partitioned into specialized processors that are integrated with general computing
- visual programming environments will have component widgets for general functions
- large grain parallelism will attract more attention than fine grain parallelism
- parallel compilers will become available for shared memory multiprocessor workstations

4. Wasilausky believed that security would not be adequately handled by commercial progress but that object mechanisms would provide the capability required within the DOD.

Industry Viewpoints

4.1 Scalable Computing 2000: Network & Nodes

Gordon Bell recommended that the C^3I architect over the next twenty years should do the following:

- define requirements and how they might evolve
- posit a framework -- stress evolvability and client programming
- minimize use of explicit parallelism -- limit to data bases, client transactions, make data bases scalable
- use the Scalable Network and Platform (SNAP) as the model (discussed below)
- use model of standards and design presented by Wasilausky for procurement

Bell described his SNAP concept of computing by 2000. (Note: the SNAP model serves as the standard expected "commercial best practice" from which the panel members made statements of their concurrence (or non-concurrence) about its suitability for C^3I systems.)

SNAP is a system that uses asynchronous transfer mode (ATM) as the underlying network with PC/Windows NT desktops as the nodes. Occasionally small multiprocessors (up to four) could be used. Desktop programming would be client-based using SQL calls to obtain data in a scalable, transparent and redundant manner. The result is that SNAP is a highly standard system and generationally scalable. A very large applications industry would provide numerous applications. All data, document, telephony and video applications could be served by SNAP. The driving force described by Bell is the continual industry rejuvenation called Moore's law. The table below summarized his plots demonstrating Moore's law for critical computing components.

TABLE A-1 Demonstrations of Moore's Law from charts presented

Technology	Factor	Growth
memory size	size in MB	x 4 every 3 years
microprocessor speed	MIPS	x 4 every 3 years (since 1990)
disk unit capacity	MB capability of 3.5" disk	x 4 every 3 years
network bandwidth	mbps of wide area network	x 1,000 within this decade

Bell's expected result is an inexpensive, 1 GMIPS PC desktop with 256 to 512 MB memory, 20 GB disks and 155 to 655 Mbps network access ports. Because of the large cost of maintenance of multiple versions of Unix a single operating system standard from a single company takes over the marketplace. Bell's choice is Windows NT. Likewise the surviving distributed virtual model is SQL. Economics of packaged software, portability, compatibility, and openness drives the market so there is a single source file, single user manual and training course, a single format for all media and arbitrary client-server operations become peer-to-peer.

4.2 *Parallel Technology for Everyman*

George Lindamood presented a concept on how broad access to information would allow citizens to more effectively interact with government. Access, awareness and understanding flowing from this access would expand the market size and improve the economics of computing. Methods for decision making using data mining and modeling and improved user interfaces and virtual reality could transfer (spin-in) to C^3I system needs. Large scale use would also provide the ideal testbed for establishing robustness and portability of models. The ubiquitous application would create opportunity for entrepreneurs and individuals could become better trained.

4.3 *Globally Scalable Architectures*

Peter Seigel presented the concept of globally scalable parallel computers. That is near linear scalability in all the capabilities needed to attack a problem -- network, data storage, and data access rates. Examples are IBM's SP2, and Convex's Exemplar. They are currently message passing but shared memory is projected. GS architectures are the basis for scalable operating environments, and applications. They are ideally suited for data mining. Applications of concern at Cornell Theory Center are data mining, visual data manipulation, data fusion, and collaboration technologies. Seigel recommended research on lightweight, high performance communication protocols, globally scalable resource managers, object-oriented models, high-level problem solving environments linked to specialized knowledgeable compilers, and algorithms that scale from many processors down to one.

4.4 *Parallel Processing for Commercial Applications*

Jeffery Mohr presented a review of the weaknesses and difficulties of MPP vendors. He pointed out that vendors have not financially succeeded in scientific and technical markets because they cannot support real science or stimulate the parallelization of major commercial engineering codes. Their competition in the business market is made up of well established players with large secure niches. Mohr pointed out the emerging competition between vendors of scientific MPPs, commercial MPPs and CMOS MVS mainframe technology. The CMOS 390 has the advantage of large legacy code already under MVS control. MVS tools on IBM's SP2 may give it an advantage. Mohr's paper pointed out that although shared memory machines -- up to 32 processors -- are now more common, independent software vendors still have trouble parallelizing their codes.

4.5 *The future of parallel software engineering in an economic context*

Jon Webb's paper provided a dichotomy to describe the economic differences between tightly coupled and distributed parallel architectures. Tightly coupled machines trade value for latency -- a computation can be completed in less time. Distributed ones trade value for bandwidth -- more instances of a given computation can be performed at the same time. Tightly coupled machines will retain importance in application areas where speed of computation is directly related to its value: weather modeling, financial analysis, program trading, and direct interaction with the real world. Networks of workstations will dominate the applications where there is no direct time value tradeoff: scientific modeling and design, graphic development, etc. Jon's concept of high time value gave the panel a cutting edge to allow it to separate C^3I applications from scientific ones.

4.6 *Mobile Network Vision*

Kim Gibson provides some insight on the extension of dynamic reconfiguration to the network. Her position is that the future in parallel computing is in geographically dispersed multiprocessors and workstations. If imagery and data could be accessed at most of the terminals, with the proper authority, the information could be available to the necessary people more quickly. By networking these geographically dispersed computing resources, Real-Time communications of imagery and data are facilitated. Because the network nodes can be anywhere, wireless communication is necessary. Forces in the most remote areas with wireless communications can receive vital intelligence and send their newest information directly without returning to a physically connected workstation.

The network will be flexible, its designers never knowing its exact configuration before deployment, so dynamic resource allocation is also a requirement. At any time during network operation, a resource may be added or deleted. The system must recognize this change and reallocate the computing resources to most efficiently perform the task. As other tasks are requested, resources must be effectively allocated to ensure efficient completion of all tasks.

In a future where communications and connectivity are highly valued, efforts must be focused on enhancing our abilities and stretching beyond the current boundaries. Networks composed of wireless, geographically dispersed computing resources with dynamic resource allocation capabilities address many of these boundaries. By connecting these resources, vital information is more available and fewer resources are necessary because of the efficient use of every node.

5. System Engineering Viewpoint

5.1 Systems Engineering of High Performance Parallel Computing Software Systems

J.C. Browne's paper described the significant future applications of High Performance Parallel Computing (HPPC) as large, complex and long-lived systems which span multiple generations of HPPC parallel architectures. This emerging generation of HPPC-based applications will be based on dynamically adaptive algorithms which are more efficient but more complex than their static predecessors. This factor alone will make the new application systems more complex than their predecessors even if they are formulated for sequential execution environments. Combining parallel structuring and complex adaptive algorithms will multiply the difficulty of developing efficient and portable HPPC application programs. Attainment of application systems that are both efficient and portable by the development processes in use for the current generation of application systems will be difficult if not intractable. Issues of systems engineering include:

- Development methods which integrate design to performance through parallel structuring with more traditional concerns for functionality and modularity
- Development methods which integrate abstract specification of HPPC execution environments
- Development methods which focus on design from components in the same mode as is done in development of hardware systems

The current HPPC software program focuses on programming languages, libraries of components, run-time systems and measurement of implemented applications but is not, as far as we know, addressing the issues cited above to any significant extent. The thesis of this informal white paper is that programming languages, libraries and run-time systems are necessary but not sufficient for the long term success of the important applications of HPPC. The systems engineering required is often discussed in three phases.

- Analysis - creates a specification (hopefully executable) of the application in application terms.
- Design - specifies the entities of the execution environment and maps the entities of the analysis phase to the entities of the execution environment to create codable entities.
- Implementation - generates the codable units defined during design.

The systems engineering needs of HPPC applications to fit in this paradigm are:

- Development methods which integrate design to performance through parallel structuring,
- Development methods which specify HPPC execution environments at design time.
- Development methods which focus on design from components in the same mode as is done for hardware design.
- Tools which implement and support the methods mentioned above.

5.2 Developing parallel applications: the Hamlet approach

Armand ten Dam's paper stated that the motivation for developing parallel applications is always to meet specific performance requirements. The means to achieve this are by developing applications very much dependent on the target hardware and by using highly machine dependent code. This results in efficient programs, but how this efficiency is achieved is another matter. Problems are to be expected when it comes to maintaining these programs or porting them to new platforms, in which case a complete rewrite of the code is not unthinkable. What is needed here is to lift the level of abstraction in program development. With respect to this we find it very important to make a clear distinction between, on one hand, designing the solution to a problem and, on the other, translating this design into an efficient and portable implementation. This idea is also shared by the partners in the Esprit project Hamlet (project 6290)*. Hamlet's main effort is put into producing a support environment for the structured design of real-time parallel applications. The applications are industrial embedded applications and are to run on transputer-based hardware. The Hamlet application development system consists of a collection of tools that can be divided into two groups: the host- based components (design tools, simulator, trace analysis tool) and the target tools (compiler/linker/loader, debugger, monitor).

5.3 Future Manufacturing Engineering Environments

Masami Uemoto reported the results of searching for appropriate engineering software tools. Her goal is the simultaneous use of heterogeneous computers in simulating manufacturing processes. The requirements were for fast and reliable communications, network transparency, heterogeneous processing, fault tolerance, data integrity, scalable processing, and portability. She found the following current limitations in parallel software tools for engineering applications:

- Developing parallel, object-oriented (OO) applications is extremely difficult. Neither adequate OO CASE tools nor OO development methodologies are currently available for parallel programming.
- Integrating parallel applications with sequential applications is not easy.
- It is difficult to develop independent parallel software modules (or objects) which can be used in a plug-and-play fashion and which can replace sequential application counterparts.
- It is difficult to debug and monitor parallel applications.
- There are no standards for parallel programming systems and software tools. Emerging distributed computing technologies and standards, such as Common Object Request Broker Architecture (CORBA), do not address parallel processing issues.
- It is crucial for long running applications to be fault tolerant against both hardware and software faults, including system crashes, network disconnect, and other abnormal situations.
- Tools are lacking in heterogeneous processing support.

5.4 Software Tools for Dynamic, Parallel/Distributed Real-Time Applications

S. Yalamanchili and Karsten Schwann are developing high performance distributed and parallel applications that simultaneously run on a variety of parallel machines linked with high performance networks. The primary purpose of their research is to exploit these machines' capabilities with highly dynamic parallel and distributed codes. Two classes of such codes are being studied: (1) real-time applications and (2) scientific codes. For Real-Time applications, dynamic behavior arises due to code complexity or, more typically, dynamic behavior in the applications' environments. For scientific applications, complex datasets often make programmers unable to anticipate program behavior.

Their group is exploring the scientific applications that offer human-interactive interfaces that can execute simultaneously with the application programs' input, computational, storage, and display tasks. For scientific applications, we envision multiple researchers or engineers working in distributed, virtual laboratories, using single or multiple coupled applications running on high performance machines and across networks. Real-time applications are used, where large-scale, complex systems executing on networks of machines are operated and controlled by multiple end users. The research agenda defined by our work is the development of tools, operating system mechanisms, and algorithms for the on-line monitoring, configuration, and control or steering of such high performance applications. For real-time systems, application adaptations address performance, timing, and reliability guarantees.

5.5 *Parallel Strategic Symbolic Decision Capability*

Carl Murphy's position was that parallel computing has great promise but is now based on specialized hardware designed to give high performance per unit cost only in a narrow applications range. This narrow applicability range increases the risk and cost of development and integration. Parallel computers exacerbate cost overruns, schedule slips, development skill base stresses, and risks of poor effectiveness. They also increase life-cycle costs if not carefully planned.

Murphy proposes that by enhancing the foundation of hardware and operating system capability that the overall cost of building applications of many types can be significantly reduced. Cost per peak performance rating may be increased, but effective in-system performance per unit cost would be significantly better. With that accomplished, the power of parallel processing will create the goal of "Strategic Symbolic Decision Computers." Meanwhile executive expectations are growing rapidly. Some of the many applications feasible with the SSDC are:

- intelligent, strategic data mining
- visualization of data base contents
- multiple DBMS sources and locations
- interactive access to raw data, information, intelligence, knowledge
- intuitive views of data
- self-adaptive pattern recognition capable data
- interactive preparation, compute, analyze cycle (a decision spreadsheet)
- powerful analytical capability for scenarios, simulation, and plan coupling and prediction

As a result of these applications, Murphy reported that industry forecasters project the need for data-intensive applications requiring a capability growth significantly higher than the uniprocessor factor of four growth in capability every three years.

This position is that the barriers to parallel computing are real and not the fault of just "software." It stresses that the problem is the lack of an adequate foundation encompassing the architecture, hardware, and operating software and a knowledgeable and supportive infrastructure. To succeed, this foundation must provide effective parallel computing for any large concurrent application, irrespective of complexity and dynamism. However, developing a knowledgeable infrastructure that can build these new applications is just as important. Technology only becomes a competitive tool when an infrastructure is able to adopt and apply them.

5.6 *Enabling Technologies*

Jay Eckard presented a point of view that information management is the key vision for C^3I. Key features are emerging architectures, large data sets, and survivability/availability. Information is the essence of C^3I, not data. Sources constantly increase in number and volume. Eckard pointed out the clash among emerging symmetric multiprocessor architectures, clustered and massively parallel, but added that solid state disks, large in-memory, radio/cellular and client/server architectures are also emerging.

6. Programming Model Viewpoints

6.1 Knowledge-based support for parallel software engineering

Douglas Smith advocated a refinement approach to software development in general, and to the engineering of parallel software systems in particular. He believes that only by starting the software development process at the requirement specification level, and refining towards various target architectures, can a truly machine-independent style of parallel programming come about. Machine-supported refinements allow engineers to factor information about software architecture, algorithms, data structures, the target architecture, particular machine, and other relevant kinds of programming knowledge into the design process. They can then achieve correct and efficient implementations. $\mathcal{Q}^2$-issues, such as predictable performance, fault tolerance, security, etc., are treated as requirements in the initial specification, and must be preserved during the refinement process. Software evolution takes place at the requirement specification level, not the code level. He gave Figure B-3 to describe software refinement.

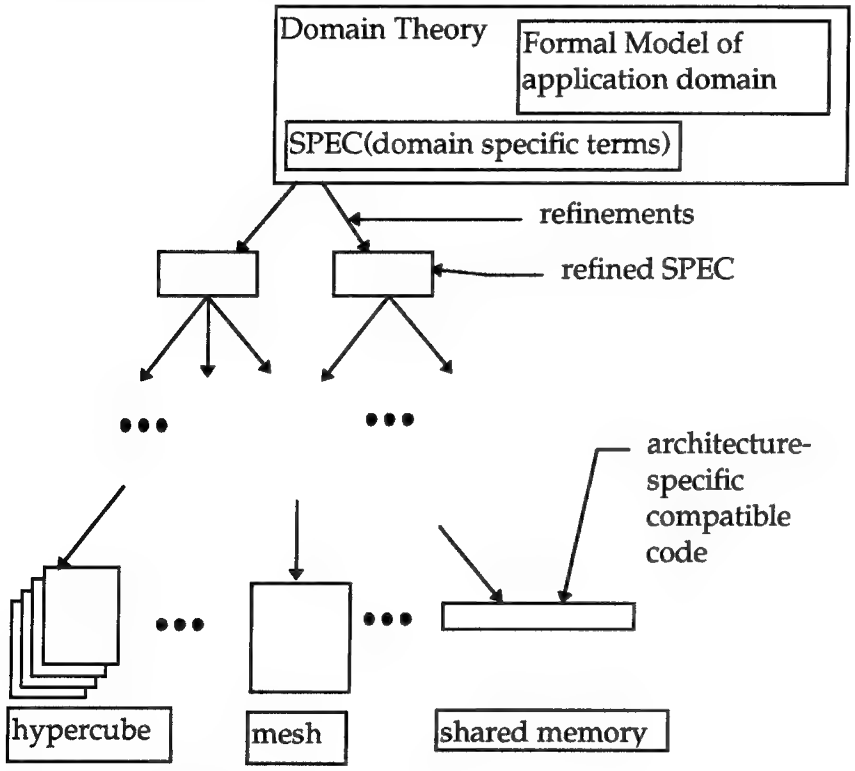

Domain Theory
Formal Model of application domain
SPEC(domain specific terms)
refinements
refined SPEC
architecture-specific compatible code
hypercube
mesh
shared memory

Figure B-3. Software refinement

A refinement-based approach to parallel software development and maintenance

Jan Prins identified the key problems faced by HPC software as: (1) the large gap between HPC design and implementation models, (2) achieving high performance for a single application on different HPC platforms, and (3) accommodating constant changes in both problem specification and target architecture as computational methods and architectures evolve.

He suggested an application development method in which high-level specifications are elaborated and refined though an iterative process. The process introduces architectural detail into a form that can be translated into efficient low-level architecture-specific programming notations. A tree-structured development process permits targeting of multiple architectures with their own flavor of implementation strategy. (See Figure B-4.) It also provides a systematic means to accommodate changes in specifications and target architectures. Prins described the Proteus system that implements these concepts plus techniques for performance assessment.

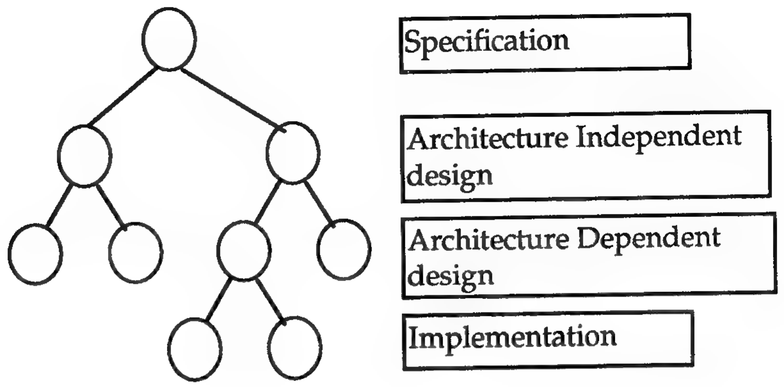

Specification
Architecture Independent design
Architecture Dependent design
Implementation

Figure B-4. Program development tree

6.2 Architecture-Independent Object-Oriented Software Development for Parallel Processing Systems

Stephen S. Yau and Doo-Hwan Bae gave an approach to identify and express parallelism in software development for parallel processing systems. They use an object-oriented paradigm which naturally reveals existing parallelism in the problem space. In addition to modifiability, maintainability and reusability, this paradigm is better than others in that the concept of an object can be used at earlier stages of the software development cycle than the implementation stage. It implies that parallel processing aspects such as parallelism and communication among parallel components can be naturally handled at the earlier stage of the software development. Consequently, it is easy for the programmers to handle parallelism and communication among parallel components. We have developed the computation model PROOF (PaRallel Object-Oriented Functional) in which the object-oriented paradigm is integrated with the functional paradigm to take advantage of the many useful features of the two paradigms.

6.3 Bulk Synchronous Parallel Computing -- A paradigm for transportable software

Thomas Cheatham presented the bulk synchronous parallel (BSP) model as a means of enabling transportable and scalable software for a wide variety of applications. In the BSP model a programmer breaks the application into a number of supersteps, each having a number of computational threads that synchronize at the end of the superstep. The model introduces the parameters g and L to deal with the costs of communication and synchronization. The parameter g represents the ratio of communication content to computational content. L represents the synchronization period to perform the completion of the superstep. Given these two parameters, the problem size and the number of processors, performance can be estimated. Cheatham and his colleagues at Harvard have developed the BSP-L programming language and compiler. It includes language constructs for BSP style, compiler, and program optimizers that include merging supersteps, efficient broadcasts, data prefetching, early put optimization and moving supersteps to earlier than specified. They have identified libraries developed by Oxford Parallel that are available on all commercial HPCs. He expects a collection of language constructs to be proposed as additions to HPFortran and HPC++, optimization technology, proofs of correctness and refinements for general purpose algorithms, a collection of provable optimal codes for a variety of target architectures and a handbook of guidelines for further use of BSP.

6.4 Definition of a Virtual Machine for C^3I Applications

Rajive Bagrodia and Mani Chandy discussed an approach of designing C^3I applications for a parallel virtual machine that is mapped to different target architectures. In this manner, the investment in C^3I software is preserved to the extent that the virtual machine hides changes between architectures and that mappings from the virtual machine to target architectures is efficient. We refer to a virtual machine for C^3I applications as a CVM.

A viable approach to the definition of a CVM is to integrate performance, real-time and fault-tolerance issues into the design process. The following must be considered:

- Computation paradigms. The CVM must be able to handle a variety of computation paradigms that are used to describe different phases of a C^3I application. The important paradigms include task parallelism, data parallelism, and vector computation. For each paradigm, the CVM must provide a set of facilities at some level of abstraction that matches the overall goal of providing a retargetable virtual machine.
- Heterogeneity. This requirement must be addressed in terms of both computing platforms and programming systems. In particular, it must support composition of multi-language and multi-paradigm programs and be targetable on the identified family of hardware platforms.
- Real-Time. To incorporate timing and resource constraints of subsystems of a C^3I application, the requirements of soft, hard, and adaptive constraints must be addressed in conjunction with the use of a variety of schedulers including priority and deadline-based, among others.
- Reliability. The level of reliability or fault-tolerance required by an application can vary significantly. The facilities provided in the CVM may range from providing eventual delivery of every message to more abstract facilities like atomic broadcasts. The cost of providing various levels of reliability must be investigated for different architectures before making a specific recommendation.
- Performance estimation. This requirement refers to the ability of a CVM to predict the performance of a specific design on a given architecture. This requirement entails a classic trade-off between precision and generality; iterative methods to support predictability on different architectures will be addressed.

7. Programming Tool Viewpoints

7.1 Virtual Shared Object Memories: A shared data foundation for parallel software engineering

Andrew Sherman and Beverly Thalberg gave their solution to producing effective responses to rapidly evolving information in a distributed, dynamic system. Their position is that this requires a data-centric, parallel and adaptive approach. The intersection of C^3I and commercial software is that it must produce robustness and reliability with long-term support and viability. They discussed the alternatives of pure shared memory and message passing. Their proposed that a content-based memory that provided virtual shared object memory had significant advantages. Their examples were data-base management systems and their own Linda and Paradise products. Their arguments are that information should be retrieved by description, not by location, that synchronization should be by information availability and that capability should scale transparently with the size and structure of the computational environment. VSOM enables many of the characteristics needed in dynamic and complex system implementation.

7.2 Component programming and standards

Mercury Computer Systems' Craig Lund described significant differences between signal processing and other C^3I applications. He explained "component programming" and outlined why it is the appropriate paradigm for signal and image processing. Mr. Lund's comments were a factor in the group's later decision to exclude signal and image processing from its deliberations.

Mr. Lund also discussed the investment "culture clash" that has developed between the COTS community and traditional defense vendors. The COTS community invests its own funds in speculative product development. In contrast, defense vendors often look to the government for help. Danger lies in unintentionally encouraging COTS vendors to emulate their defense cousins.

7.3 Parallel software development: future of parallel and distributed software development tools

David Rich provided this input. Much of the discussion argued that C^3I hardware needs will be met by networks of COTS resources. There was a suggestion that COTS software tools will also be able to meet the needs of C^3I. In general, I agree with these ideas, however I think it is important to note that the C^3I community will likely wish to field some of the largest applications to be implemented on these COTS solutions. This is a non-trivial point as many systems may not function correctly when pushed to the limit.

Tools to simulate, monitor, tune and maintain large software systems may have to be built as they may not be provided by commercial vendors who are primarily interested in smaller systems. We believe that in the future an "application" will not reside on a single processor. In fact, our definition of the future application platform includes much more than a given parallel processor. Our picture of the platform of the future is a network of computing resources which might include the following types of machines; clients (Unix workstations or PCs), servers (parallel processors with Unix or perhaps Windows NT) and special-purpose devices including embedded systems, compute engines, I/O processors, network devices or application monitoring equipment.

They believe the programmer needs a way to debug his whole application from within one development context. Given that the "platform" is a collection of networked devices (including parallel machines), this implies that the developer's tools must function on all architectures. In addition, the tools must be able to address all architectures simultaneously.

8. System Effectiveness

8.1 Parallel software must support efficient execution of general applications

Min-You Wu provided the following:

"While regular applications can be solved efficiently, only a small fraction of applications are regular. One of the major problems in parallel computing is that the current technique cannot solve applications with irregular computation and communication structures successfully. Irregular applications account for a large portion of real problems. Many applications in C^3I *systems fall into this category."*

An irregular problem is difficult to solve on parallel machines because of its irregular data domain and irregular computation structure. Only with a run-time support system can an irregular application be solved efficiently. A real-time, run-time system is crucial for C^3I applications.

8.2 ADAM: Application Driven Architecture Methodology

Martin Davis provided a summary of his position.

"Much is made about creating 'architecture- independent' tools and 'architectural independent' software representations. Such tools and representations are necessary for the portability and ease of creation of software. However, when it comes to predicting the performance of a particular application on a particular architecture and its specific platform (i.e., the hardware and systems software instantiation), one cannot ignore the architecture nor the specific instantiation. Thus, it is important to understand how to map the architecture-independent tools and representations to specific architectures and platforms in order to understand whether the performance is sufficient for the real-time C^3I *requirements."*

8.3 Tools for Parallel Software Technology

Sanjay Bhansali and C. S. Raghavendra believed that the need for parallel computing is, and will continue, growing in the future, as we solve problems of ever-increasing scale and complexity. Instead of expensive parallel computers, users will buy clusters of workstations and personal computers for a fraction of the price. The problems of data distribution, synchronization, etc. will still have to be solved for running distributed applications in these environments. But there will be a stronger need than ever for writing effective distributed applications efficiently. Our position is that we need to create domain-specific tools that will help users create parallel programs in specific application areas. These tools will sacrifice generality in order to provide more powerful help in well-defined application domains. They reported that they use a powerful algorithm level analysis of sequential programs followed by a semantic-preserving transformation of sequential code into parallel code. These reverse engineering methods have already shown promise in maintaining legacy code. By extracting parallelism at a higher level of abstraction, they expect two benefits:

- a deeper understanding of the sequential program to help identify parts of the program to be parallelized, as well as parts that should not be parallelized;
- ability to replace blocks of sequential code with highly optimized parallel code, possibly from a code library developed for a target machine, to yield significant speedups.

8.4 Flexible and Efficient Abstractions for Parallel Programs

Matt Rosing's position paper advocated new abstraction mechanisms based on compile time language constructs specialized to a fairly narrow domain. He compared his approach to with object-oriented and library approaches. He believed that oriented technology is very flexible but has severe tradeoff difficulties between efficiency and flexibility. He believed that libraries are context insensitive. Rosing advocated the domain-specific compiler abstraction because a compiler can examine the entire context of how a construct is used in order to generate efficient code from user-defined, compile-time abstractions. He has implemented distributed arrays as an object whose characteristics are defined by a "mapping function." A mapping function is essentially a type that consists of a set of user provided code segments. It has proven most useful for low level abstractions in scientific computations that will be executed a large number of times at run time and take little time per instantiation. An example is index calculations. Rosing expects that commercial and C^3I applications will have large distributed data structures but probably not arrays. There will be added complexity with embedded and heterogeneous systems. Parallel I/O will also benefit from this technology.

8.5 Issues in the Specification and Design of Parallel Programs

Robert Rabb's paper presented an overview of the main specification and design issues for parallel systems of programs from a software engineering perspective. The main issues considered are portability, (hierarchical) design correctness, and reliability of implementations. A parallel system design approach based on the Large-Grain Data FlowJ2 (LGDF2) computation model is outlined. LGDF2 extends the safety of functional-style programming to include networks of processes whose run-time communication and synchronization patterns can be constrained at "design-time" to yield the desired safety, reliability and programmability for parallel programs at run-time.. An assessment of LGDF2 as the basis for unified specification, design, and implementation of parallel programs is given, along with a brief assessment of its potential impact on parallel software development and software project management. The approach lends itself to better visibility into the current state of evolving system designs, and can speed development of large, complex, yet reliable parallel systems via graceful evolution of rapid parallel system prototypes.

With appropriate tool support, we see this technology as having potentially as large an impact on parallel software engineering for commercial and government applications as WYSIWYG editors and PostScript printers have had on document production. The key similarity is that novice users can be productive without having to understand the details of what makes either technology "work."

☆U.S. GOVERNMENT PRINTING OFFICE: 1995-710-126-20068

MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.